

SYSTEM JOURNAL

ISSN 0933-9434 | September/Oktober 1988 | 2.Jg./Heft 5
ÖS 150,- | DM 19,80 | sfr 19,80



MS-DOS 4.0:

Ein neues, freundliches DOS

Die Geschichte von MS-DOS

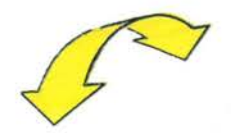
Große Zeichensatztabelle für MS-DOS und Windows

Schwerpunkt Windows:

Die Datenbankoberfläche SQLWindows

Utilities und Routinen für Windows

Debuggen unter Windows mit CodeView oder Utilities



Im Mittelpunkt steht der Mensch.



Konzentration durch Entspannung

Alfaskop.

Möchten Sie sich nach längerer Bildschirmarbeit noch wohlfühlen? Und die Technik auf den Menschen abgestimmt wissen, nicht umgekehrt? Dann sind Sie ergonomiebewußt* und bei Nokio Data genau richtig: Terminals, PCs, Workstations und Netzwerke dokumentieren beispielhaft, wie man den Menschen, seine Arbeit und seine Gesundheit in den Mittelpunkt rückt.

Alfaskop ist das Dialog- und Netzwerk-Konzept von Nokia Data – ergonomisch aus Tradition, zukunftssicher und wirtschaftlich durch moderne Technik. Vom führenden europäischen Hersteller.
Mehr Sicherheit für Ihre Investitionen gibt es nicht.

**Wenn Sie daran interessiert sind, was wir unter Ergonomie verstehen, fordern Sie einfach unsere kostenlose „Kleine Ergonomie-Fibel" an!*

Nokia Data ist der Spezialist für ergonomische Terminals, Workstations und Netzwerk-Konzepte. Als Tochter des skandinavischen Nokia-Konzerns zählt das Unternehmen mit seinen 8000 Mitarbeitern und einem Umsatz von über 2 Milliarden Mark zu den großen und etablierten Anbietern auf dem europäischen DV-Markt.





Unser Maßstab ist der Mensch.

Nokia Data GmbH D 4000 Düsseldorf 30, Vogelsanger Weg 91, Tel. 0211/6109-0
.**B**remen 0421/498207 .**F**rankfurt 06104/6902-0 .**H**amburg 040/540008-0 .**H**annover 0511/6706-1 .**M**annheim 0621/28947
.**M**ünchen 089/50078-0 .**N**ürnberg 0911/633095 .**S**tuttgart 0711/72009-0 NE.W A 13.8

Ausgabe September/Oktober 1988




Microsoft Windows

Die Datenbankoberfläche SQLWindows 4

Microsoft Windows als Basis einer einzigartigen Entwicklungsumgebung: SQLWindows bietet Programmierern die Möglichkeit, Dateneingabe- und -manipulationswerkzeuge zu erstellen, die die ausgefeilte Benutzerschnittstelle von Microsoft Windows mit der Leistungsfähigkeit und Universalität von SQL kombinieren.

Screen-Save-Funktion für Windows 10

Eine Systemutility für Windows: In diesem Artikel wird eine Anwendung für Windows entwickelt, die nach einer einstellbaren Zeit des »Nichtstuns« am Rechner (keine Tastatur- oder Mauseingabe) den Bildschirm dunkel schaltet, bis wieder eine Eingabeaktion stattfindet.

Dynamische Dialogboxen unter Windows 26

Dialoge ganz nebenbei: In Microsoft Windows 2.0 ist es möglich, Dialogboxen oder Eingabemasken dynamisch, so ganz »nebenbei« zu erzeugen. Wir zeigen, wie's geht.

Debuggen unter Microsoft Windows

Windows-Programme effektiver debuggen 60

Eine einfache Utility erleichtert die Windows-Programmierung: Unglücklicherweise funktionieren viele der traditionellen Test-Techniken in der Windows-Umgebung nicht, doch es gibt äquivalente Möglichkeiten.

Ein interaktives Debugging-System für Windows 76

Bald gibt es den interaktiven Debugger CodeView auch für Microsoft Windows. Wir haben uns eine Beta-Version ausführlich angesehen.

MS-DOS 4.0

Ein neues, freundliches DOS 36

Microsoft hat soeben die Version 4.0 von MS-DOS angekündigt. Wir beschreiben die Neuheiten, deren wichtigste die bildschirmorientierte Benutzerschnittstelle ist.

Geschichte und Aufbau von MS-DOS 51

Still going strong: Nach acht Jahren und zahlreichen Versionen ist MS-DOS in der Version 4.0 so komfortabel und umfangreich wie nie zuvor. Anlaß für uns, uns einmal ausführlich mit der Entwicklungsgeschichte von MS-DOS von 1980 bis heute zu beschäftigen.

Makro-Assembler

Verbesserte Makros und Sprachenschnittstellen 85

Die Version 5.1 des Microsoft Makro-Assemblers bietet viele neue Möglichkeiten, die die Programmierung in Assembler bedeutend vereinfachen.

Rubriken

Mitteilungen .... 54 — Neue Produkte, Aktuelles.
Termine .... 71 — Die Termine des Microsoft-Instituts.
Fragen & Antworten .... 74 — Tips zu Microsoft Windows.
Buchbesprechung .... 42 — Die PC-Referenz für Programmierer.
Buchauszug .... 43 — Nützliche Tabellen für die Programmierung.
Zeichensatztabelle .... 46 — Große Übersicht: Die Zeichensätze von MS-DOS und Windows.
Impressum 90
Inserentenverzeichnis 71

Microsoft Windows als Basis einer einzigartigen Entwicklungsumgebung:

Die Datenbankoberfläche SQLWindows

Die Firma Gupta Technologies aus Menlo Park, Kalifornien, ein Anbieter von Datenbanksoftware, hat vor kurzem eine einzigartige Windows-Anwendung mit dem Namen SQLWindows vorgestellt. Dabei handelt es sich um ein Anwendungsentwicklungssystem, das auf Datenbanken spezialisiert ist, die mit der Abfragesprache SQL (structured query language) arbeiten. Es wurde entwickelt, um Programmierern die Möglichkeit zu bieten, Dateneingabe- und -manipulationswerkzeuge zu erstellen, die die ausgefeilte Benutzerschnittstelle von Microsoft Windows mit der Leistungsfähigkeit und Universalität von SQL kombinieren.

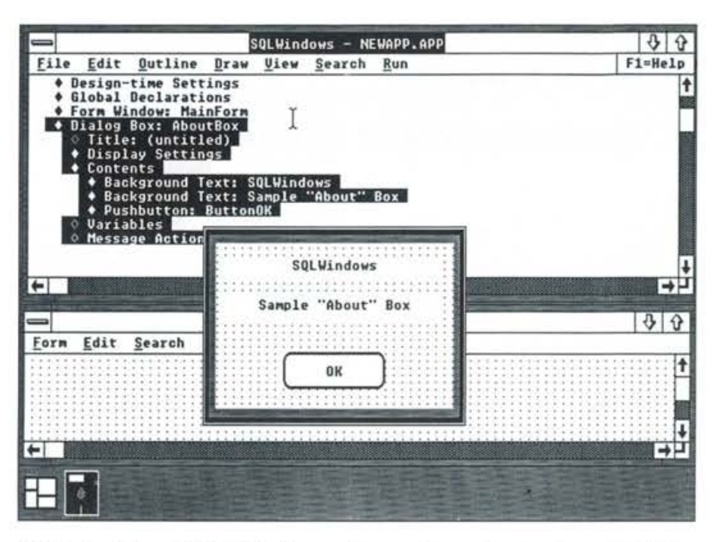


Bild 1: *Eine SQLWindows-Anwendung im ersten Stadium der Entwicklung. Es soll eine About-Box mit einer OK-Taste definiert werden.*

SQLWindows ist das dritte Glied in einer Reihe von Datenverwaltungsprodukten, die Gupta Technologies das SQL-System nennt. Die beiden anderen Komponenten sind SQLBase, ein Datenbankserver für 80286/386-Umgebungen und SQLNet, eine transparente Brücke zu IBMs DB2. (Zukünftige Versionen werden auch Verbindungen zu anderen Großrechnern-DBMS bieten.)

Gupta Technologies wurde 1984 von Umang P. Gupta und D. Bruce Scott gegründet, die vorher in Schlüsselstellungen bei Oracle beschäftigt waren. Gupta war der Vizepräsident und Leiter der Mikrocomputerprodukte-Abteilung bei Oracle und Scott ist ein Co-Autor des Oracle Datenbanksystems.

Das erste Produkt der Firma, SQLBase, sorgte gleich für Schlagzeilen, als es 1986 vorgestellt wurde. Es war der erste echte Datenbankserver (im Gegensatz zu einem Multiuser-DBMS, das auf einem Fileserver läuft), der speziell für Mikrocomputer gedacht war, die mit Microsoft Networks arbeiten. Eine erweiterte Version dieses Produktes zog kürzlich die Aufmerksamkeit der Öffentlichkeit auf sich, als Lotus Development ankündigte, das dies die Basis für das zukünftige Lotus/DBMS-Produkt sei.

Lotus wird natürlich seine eigenen Anwendungsentwicklungswerkzeuge für Lotus/DBMS anbieten. Da das Lotus-System jedoch im Prinzip das gleiche Produkt ist wie SQLBase und auch das gleiche API (application programming interface - Anwendungsprogrammierschnittstelle) enthält, laufen alle Anwendungen, die in irgendeiner Sprache für SQLBase geschrieben wurden, ohne Änderungen auch unter Lotus/DBMS. Dazu gehören auch solche Anwendungen, die mit Guptas SQLWindows entwickelt wurden.

»SQLWindows arbeitet heute mit SQLBase und in Zukunft mit dem Lotus/DBMS-Server«, bestätigt Umang Gupta. »In Zukunft werden wir auch ähnliche Unterstützung für andere Datenbankserver anbietet. Von Anfang an ist es ein Entwicklungsziel gewesen, daß das Produkt mit anderen SQL-Systemen zusammenarbeiten kann.«

Gupta weist audrücklich darauf hin, daß der angekündigte SQLServer von Microsoft und Ashton-Tate zu den Produkten gehört, die unterstützt werden sollen. (Über den SQL-Server hat das Microsoft System Journal bereits ausführlich in der Ausgabe März/April 1988 berichtet.)

»SQLWindows ist einzigartig«, fährt Gupta fort. »Es gibt auf dem Markt keinen Anbieter, der ein ähnliches Produkt hat oder auch nur angekündigt hat, das auf den Presentation Manager und Windows zugeschnitten, systemunabhängig und sowohl in die Fensterumgebung als auch in SQL eingebunden ist.«

Eine Version von SQLWindows für den Presentation Manager ist auch in Arbeit; Gupta glaubt, daß sie im vierten Quartal dieses Jahres fertig sein wird. Danach plant die Firma, sich auf den Macintosh zu konzentrieren.

»Unsere langfristigen Pläne«, sagt Gupta, »gehen davon aus, daß die Zielsysteme PC und PS/2 unter dem Presentation Manager oder Windows, der Macintosh und UNIX-Workstations unter einem standardisierten UNIX-Fenstersystem sind.« Die Ausrichtung auf so viele Systeme bei gleichzeitiger Kompatibilität mit einer wachsenden Anzahl von SQL-Systemen auf dem Markt und gleichzeitig die Erweiterung von SQLNet um Verbindungsmöglichkeiten zu anderen Großrechner-DBMS-Systemen, ist, man kann es nicht anders sagen, ein ehrgeiziges Programm. Gupta Technologies beschäftig momentan etwa 25 Leute, davon 12 Vollzeit-Programmierer. »Wir werden unsere Entwicklungs- und Vertriebsmannschaft bald erheblich vergrößern«, meint Gupta dazu.

Interaktive Fenster

Gupta und Michael Geary, der Chefentwickler von SQLWindows, ermöglichten dem Microsoft System Journal einen Blick auf eine Beta-Version ihres neuesten Produkts.

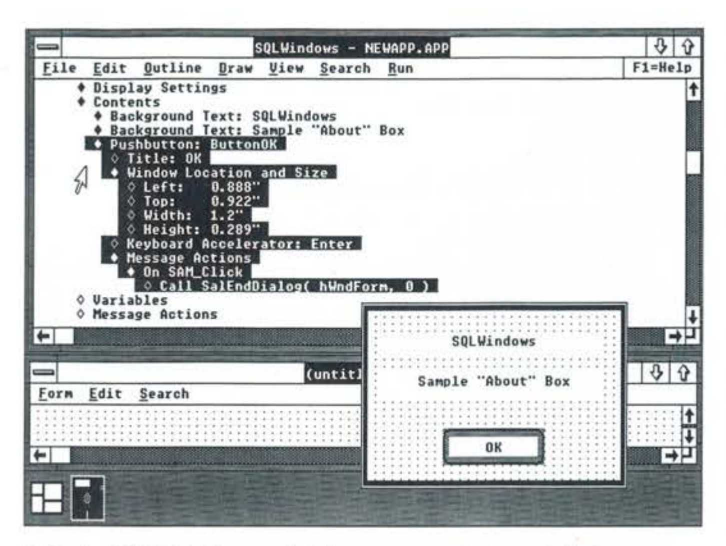


Bild 2: *SQLWindows wird dazu verwendet, zu definieren, was passiert, wenn der Benutzer »OK« anklickt. Hier soll einfach nur zum übergeordneten Fenster zurückgekehrt werden.*

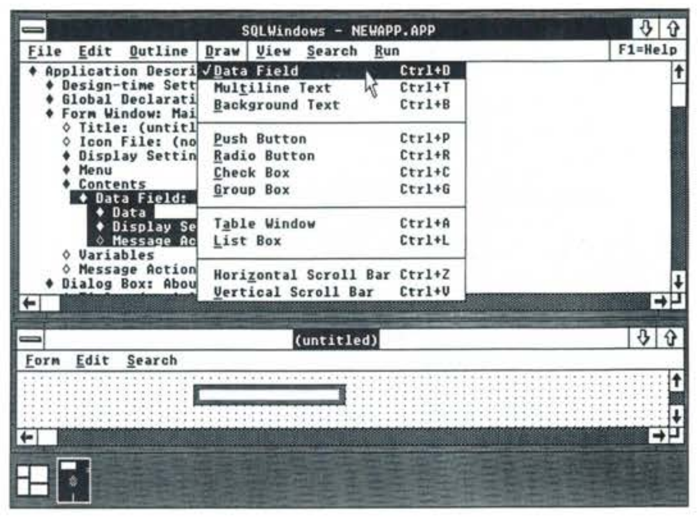


Bild 3: *Ein Datenfeld wird mit dem Draw-Befehl von SQL-Windows zum Layoutfenster hinzugefügt. Änderungen im Layoutfenster werden sofort im Gliederungsfenster dargestellt.*

SQLWindows besteht aus drei Hauptkomponenten: einem visuellen Maskenlayouteditor (ähnlich wie der Dialog-Editor aus dem Microsoft Windows Software Development Kit), einem Gliederungseditor und einer Programmiersprache mit dem Namen SQLWindows Applications Language (SAL). Gliederungs- und der Layouteditor arbeiten zusammen: Änderungen, die in einem von beiden Fenstern gemacht werden, bewirken eine sofortige Aktualisierung des anderen Fensters.

Bild 1 zeigt eine Beispielanwendung unter SQLWindows in einem sehr frühen Stadium der Entwicklung. Der Programmierer hat eine einfache About-Box angelegt. Die Box wird im Layouteditor (dem unteren Fenster in Bild 1) angezeigt, während ein Teil des entsprechenden Quellcodes ausgewählt im Gliederungsfenster zu sehen ist.

Die ausgefüllten Rauten in der Gliederung kennzeichnen Überschriften, die untergeordnete Einträge haben; die nicht gefüllten Rauten kennzeichnen Einträge, die keine untergeordneten Einträge haben. Bild 1 zeigt den Eintrag »Contents«, der »Dialog Box: AboutBox« untergeordnet ist und so expandiert wurde, daß zwei Zeilen Text und eine Schaltfläche zu sehen sind.

In *Bild 2* ist ein anderer Auschnitt aus dem Quellcode für die gleiche About-Box zu sehen. Hier geht es um die Schaltfläche; es sind der Titel, die Position, die Größenattribute, ihre »Acceleratoren« (die Tasten, die die gleichen Operationen ausführen, wie das Anklicken der Fläche) und die »Meldungsaktionen« dieser Schaltfläche zu sehen. Im Zusammenhang mit dieser Anwendung ist nur ein Ereignis von Interesse (der Benutzer klickt die Fläche an oder betätigt die entsprechende Taste). Dieses Objekt benötigt nur eine einzige Meldungsaktion. Der Eintrag unter »Message Actions« besagt: »Wenn der Benutzer klickt, beende den Dialog und kehre zum übergeordneten Fenster (parent window) zurück«.

`SAM_Click` bedeutet, daß die Schaltfläche angeklickt wurde; `SalEndDialog` und `SalParentwindow` sind Funktionen der SQLWindows Application Language.

Schrittweise Enthüllungen

Gupta hebt hervor, daß SQLWindows für die Bedürfnisse von Programmierern entwickelt wurde und die Gliederung des Quellcodes diesem Entwicklungsziel gerecht wird.

»Viele 4GLs (fourth generation language - Sprache der vierten Generation) gehen mit Dialogboxen an das Schreiben von Programmen heran«, sagt er. »Man fügt irgendwo ein Feld hinzu und sofort erscheint eine Dialogbox und stellt einem alle möglichen Fragen.« Der Nachteil dieses Arbeitsstils liegt nach Guptas Meinung darin, daß man das Programm nicht sehen kann, während es entwickelt wird, da es überall in Dialogboxen verborgen ist. »Das mag für einen Endbenutzer richtig sein, denn diese wollen meistens nicht viel vom Programm sehen. Programmierer müssen jedoch - zumindest manchmal - alles einsehen können. Man muß ihm die Möglichkeit dazu bieten, eine Art von schrittweiser Enthüllung. Eine Gliederung ist dafür das passende Mittel.«

Der Entwickler Geary sagt, daß die Idee, einen Gliederungseditor in SQLWindows einzubauen, ihm kam, als er die Dokumentation zu einem anderen Anwendungsentwicklungssystem studierte, das nicht auf Gliederungen basierte. »Ich bemerkte in vielen Fällen, daß sie in der Dokumentation eine Gliederung zeichneten um zu erkären, was sie taten. Und ich dachte, wenn sie eine Gliederung aufzeichnen, um zu erklären, was sie tun, dann kann man doch gleich eine Gliederung nehmen um das zu tun, was man tun will.« Ein Programmierer, der SQLWindows verwendet, kann sich vollkommen frei zwischen dem Gliederungsfenster und dem Layoutfenster bewegen, je nachdem, welches gerade günstiger ist.

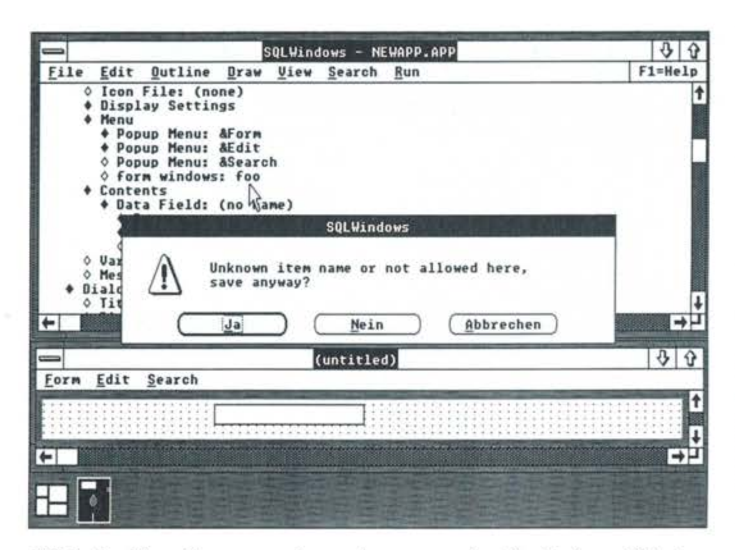


Bild 4: *Der Programmierer kann auch direkt im Gliederungsmodus arbeiten. Änderungen im Gliederungsfenster werden sofort im Layoutfenster dargestellt.*

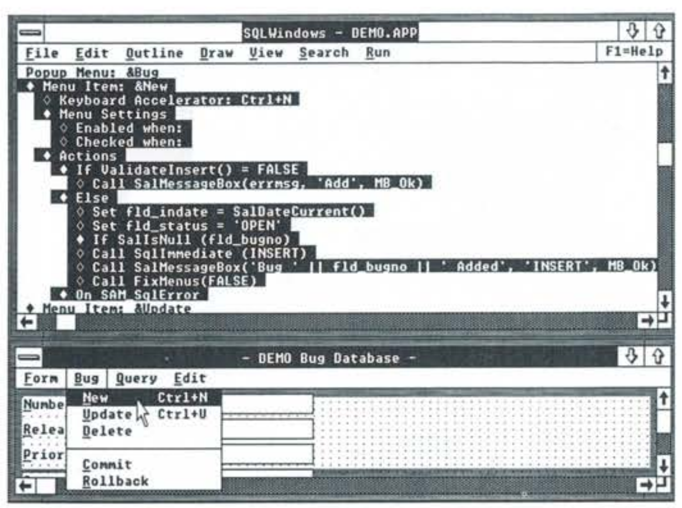


Bild 5: *Der Programmierer kann SQL-Anweisungen direkt im Gliederungsfenster einfügen. Die Anweisung* `SQLImmediate` *führt SQL-Anweisungen direkt und sofort aus.*

Die beiden Fenster sind ganz einfach zwei Darstellungen einer gemeinsamen Datenstruktur; Änderungen in einem von beiden verändern die zugrundeliegende Datenstruktur, was sich sofort im anderen Fenster wiederspiegelt.

Die Bilder 3 und 4 illustrieren diese beiden Arbeitsweisen. In Bild 3 fügt der Programmierer gerade ein neues Datenfeld in das Layoutfenster ein, indem er die Werkzeuge aus dem Draw-Menü von SQLWindows verwendet. In Bild 4 hat der Programmierer direkt in der Gliederung eine Änderung gemacht, indem er die [Ins]-Taste gedrückt und einfach geschrieben hat.

Die Änderung, die der Programmierer hier durchgeführt hat, ist hier nicht möglich. Ein Maskenfenster kann nicht in ein Menü eingefügt werden. Der Programmierer hat drei Möglichkeiten: zur Gliederung zurückgehen und den Eintrag korrigieren ([Esc]), den fehlerhaften Eintrag ganz herausnehmen (»No«) oder den Eintrag so zu übernehmen (»Yes«). Wenn »Yes« gewählt wird, läßt SQLWindows den Eintrag an der Stelle, wandelt ihn jedoch in einen Kommentar um, so daß sich der Programmierer später um dieses Problem kümmern kann.

Zugriff auf SQL

Der ganze Zugriff auf SQL wird über 15 Funktionen abgewickelt, die SQLWindows zur Verfügung stellt. Dazu gehören: `SQLImmediate`, womit eine SQL-Anweisung sofort ausgeführt wird; `SQLPrepare`, womit eine SQL-Anweisung für die spätere Ausführung vorbereitet wird; `SQLConnect`, das die Anwendung mit einer bestimmten Datenbank verbindet; `SQLDisconnect`; `SQLFetch`, mit dem eine Datenzeile gelesen wird; `SQLFetchNext` und `SQLFetchPrevious`. In *Bild 5* ist die Verwendung der Funktion `SQLImmediate` zu sehen.

Bei der Anwendung in Bild 5 handelt es sich um eine Datenbank, die Geary dazu verwendet hat, um Fehler in den Vorversionen von SQLWindows zu notieren. *Bild 6* zeigt eine ausgefüllte Maske als Teil dieser Datenbank. Bei dem ausgewählten Programmbereich handelt es sich um den Teil, der bei dem Befehl »New« aus dem Menü »Bug« ausgeführt wird. Zunächst wird eine Überprüfung vorgenommen (`ValidateInsert`). Wenn diese negativ ausfällt, wird eine SAL-Meldungsbox aufgerufen. Sonst wird die Funktion `SQLImmediate` zweimal aufgerufen - beim ersten Mal, um die Nummer des nächsten Bugs in sequentieller Reihenfolge festzustellen (`select(max(bugno)+1)`, und im zweiten Fall, um den Datensatz einzufügen.

Gupta weist ausdrücklich darauf hin, daß der SQLWindows-Programmierer eine explizite Kontrolle über alle SQL-Anweisungen hat, die das System erzeugt. »In anderen 4GLs«, meint er, »werden SQL-Anweisungen implizit erzeugt, der Programmierer oder Benutzer führt SQL-Anweisungen nicht direkt aus. In dem Maße, wie das der Fall ist, hat der Programmierer weniger oder gar keinen Einfluß mehr. In SQLWindows können Sie SQL-Anweisungen ganz nach Belieben ausführen, jederzeit und in beliebiger Reihenfolge.«

Interne und externe Funktionen

Der Aufruf von `ValidateInsert` in Bild 5 ist ein Beispiel für die »internen Funktionen« von SQLWindows. Das sind Funktionen, die nicht zum Repertoire der ins System eingebauten Funktionen gehören, sondern vom Programmierer mit SAL- und SQL-Funktionen ganz nach den Erfordernissen programmiert werden können. Das System sieht auch externe Funktionen vor, die sich entweder in dynamischen Linkbibliotheken oder in statischen C-Bibliotheken befinden können.

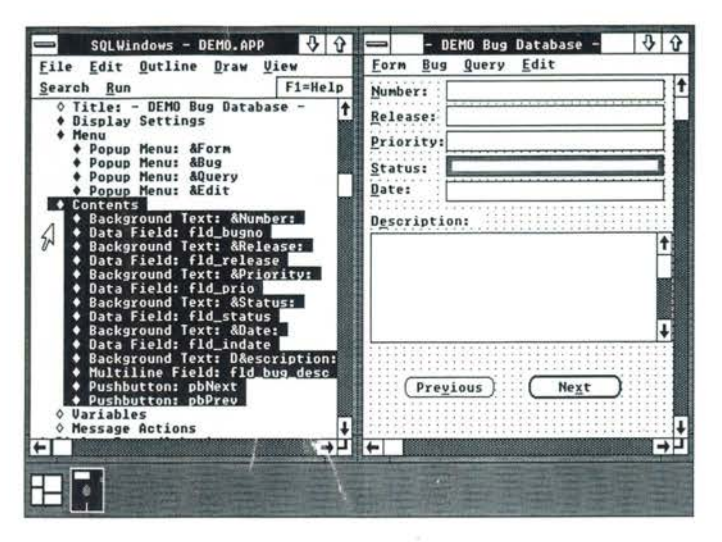


Bild 6: *Ein Beispiel für eine fertige Eingabemaske, im Layoutfenster zu sehen, und die zugrundeliegende Gliederung, zu sehen im Gliederungsfenster.*

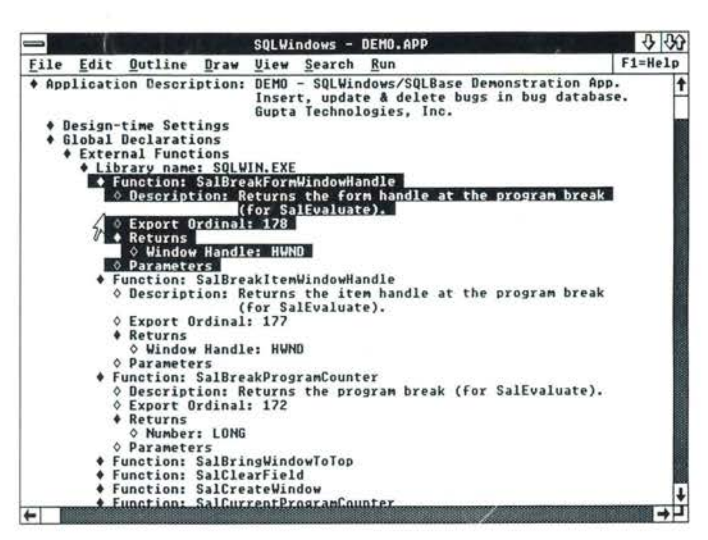


Bild 7: *Der Programmierer kann zu der SQLWindows-Umgebung eigene Funktionen hinzufügen. Hier werden z.B. Funktionen aus der externen Bibliothek* `SQLWIN.EXE` *aufgerufen.*

Bild 7 zeigt die Deklaration von Funktionen, die in zwei externen Bibliotheken gespeichert sind: `SQLWIN.EXE` und `USER.EXE`. Letztere ist Bestandteil des Windows-Toolkits. Bild 8 zeigt den Programmcode für eine interne Funktion mit dem Namen `DoQuery`, die SQL-Funktionen verwendet, um eine Datenzeile vorzubereiten und zu lesen.

Warum Windows?

Warum wählte Gupta Technologies Windows als Oberfläche für SQLWindows? Wenn man den Entwickler und den Geschäftsführer fragt, erhält man zwei Varianten derselben Antwort.

Geary meint: »Windows hat die Benutzeroberfläche, die ich mag. Es ist wirklich darauf zugeschnitten, Anwendungen zu entwickeln, die leicht zu erlernen und dann auch leicht zu verwenden sind, wenn man sie erlernt hat. Windows bietet uns auch Tools, die wir an unsere SQLWindows-Entwickler weitergeben können, die wiederum damit gute Anwendungen mit einer sauberen Oberfläche entwickeln können.«

Gupta sagt dazu: »Eine wesentliche Grundlage unseres Geschäfts ist, daß Benutzer einen einheitlichen Zugriff auf persönliche Daten, Daten der Abteilung und der ganzen Firma haben möchten, und daß SQL die lingua franca ist, die diesen Zugriff ermöglicht. Die zweite Grundlage ist, daß der Zugriff auf Datenbankserver von ausgefeilten Anwendungen aus stattfinden wird, die geschrieben werden, um die Vorteile der nächsten PC-Generation zu nutzen. Und das bedeutet, von dem OS/2 Presentation Manager und Microsoft Windows - nicht von dummen Terminals.«

Die Technologie für »industrie-starkes Datenmanagement«, wie Gupta es nennt, ändert sich derzeit gewaltig. SQL wird zum Standardmedium für den Datenbankzugriff. Und die Verfügbarkeit von 80386-PCs und lokalen Netzwerken mit hoher Geschwindigkeit hat ermöglicht, daß Datenbankserver auf PCs für den Einsatz in Abteilungen genauso kosteneffektiv sind wie Systeme auf Minicomuter-Basis.

Gupta Technologies hat einen guten Riecher für diese Entwicklungsrichtung bewiesen. Sie richtete sich schon früh auf SQL aus und entwickelte den ersten Datenbankserver für PCs.

Auf der Bedienungsseite haben die Fortschritte in der Beutzerschnittstellentechnologie den langsamen Anfang vom Ende traditioneller, auf Minicomputern basierenden, textorientierten 4GLs signalisiert. In den nächsten ein oder zwei Jahren können wir damit rechnen, Myriaden neuer Ansätze zur Anwendungsentwicklung von Softwarefirmen zu sehen, die ihre Wurzeln in der Microcomputer-Welt haben. OS/2 und der Presentation Manager werden eine wichtige Rolle bei dieser Entwicklung spielen. Durch die Verwendung der heute verfügbaren Windows-Technologie ist Gupta mit SQLWindows ein Trendsetter in der Presentation Manager- und OS/2-Umgebung der nicht mehr allzu fernen Zukunft.

Microsoft Windows intern

Michael Geary, der Entwickler von SQLWindows, mag die Programmierung unter Microsoft Windows sehr.

»Am wichtigsten«, sagt er, »ist die Benutzerschnittstelle. Eine der schönen Dinge an Windows ist, daß der Windows Style Guide sehr detailliert eine ausgezeichnete Benutzerschittstelle beschreibt. Das macht das Leben für einen Programmierer wesentliche einfacher und es macht es auch für den Benutzer leichter. Wenn jemand einmal eine Anwendung erlernt hat, ist es für ihn sehr viel einfacher, eine andere zu erlernen.«



Bild 8: *Der Programmierer kann eigene Funktionen mit SQLWindows SAL- und SQL-Funktionen entwickeln. Hier ist Beispiel für eine interne Funktion (*`DoQuery`*) zu sehen.*



Bild 9: *Mit SQLWindows kann der Benutzer auch Anwendungen entwickeln, die nicht auf SQL basieren. In diesem Beispiel hat der Anwender einen Taschenrechner entwickelt.*

Geary schätzt die Tatsache, daß Windows auch die Tools bietet, um diese Schnittstelle aufzubauen. »Wenn ich ein Menü schreiben will, brauche ich kein Programm zu schreiben, das weiß, wie Menüs funktionieren. Alle Grundlagen und Feinheiten der Benutzerschnittstelle sind bereits eingebaut.«

Meldungen und Unterfenster

Geary, der sehr viel sowohl auf dem Macintosh als auch in der MS-DOS-Umgebung programmiert hat, bevor er sich an SQLWindows machte (zu seinen Verdiensten gehört auch das vielgeschätzte Kommunikationsprogramm Transend/PC), war besonders von zwei Windows-Aspekten beeindruckt: Meldungen und Unterfenster.

»Ich werde oft aufgefordert, den Macintosh und Windows miteinander zu vergleichen«, sagt er. »Vom Standpunkt des Benutzers und dem des Programmierers sind sie sehr ähnlich. Doch mich hat es sehr beeindruckt, wie Windows die Macintosh-Vorstellung von Ereignissen genommen hat und sie zum Meldungskonzept verallgemeinert hat. Meldungen sind wie eine Übermenge der Ereignisse. Was man in Windows auch machen kann und auf dem Mac nicht geht, ist, daß man seine eigenen Meldungen erstellen und von einer Anwendung zur anderen schicken kann. Das hat DDE (dynamic data exchange) ermöglicht; DDE ist einfach nur ein Protokoll für den Austausch von Meldungen.«

Genauso bei den Unterfenstern; es gibt im Prinzip nur eine Fensterart auf dem Macintosh, erläutert Geary. Jedes Fenster ist allen anderen gleichgestellt, und wenn ein Programm ein Fenster aufgebaut hat, ist es Sache des Programmierers, alles zu managen, was darin vor sich geht. Der Mac bietet auch Tools wie Push- und Radiobuttons, Scrollbalken uns so weiter, doch ein Programm muß die Details verwalten, wo sie stehen und wie sie verwendet werden.

In Windows sind die gewöhnlichen Objekte, soweit es die Interaktion des Benutzers mit der Anwendung angeht, zum Beispiel die verschiedenen Bestandteile einer Dialogbox, typischerweise als Unterfenster implementiert. Und da Unterfenster im Prinzip ganz genauso funktionieren wie Hauptfenster, passieren eine Menge Dinge automatisch, die auf dem Macintosh explizite Programmierung erfordert hätten.

Unterklassen

»Was ich noch an dem Unterfensterkonzept mag,« fügt Geary hinzu, »ist, daß Windows einem eine Reihe von vordefinierten Klassen von Unterfenstern zur Verfügung stellt und es dann einfach macht, sie unter einem Fenster anzuordnen. Man nimmt einfach eine vordefinierte Klasse und fügt ein wenig eigenes Verhalten hinzu, indem man seine eigenen Fensterfunktionen vor die schiebt, die es bereits gibt und so effektiv die Meldungen herausfiltern kann, die man anders behandeln möchte.«

Geary hat diese Technik bei der Programmierung von SQLWindows sehr viel verwendet. Objekte im Layouteditor müssen anders reagieren, wenn der Benutzer von SQLWindows eine Anwendung entwickelt als wenn der Endbenutzer mit der Anwendung arbeitet. Der Entwickler muß auf solche Objekte wie Pushbuttons klicken können, um die Größe zu verändern oder sie zu verschieben; beim Endbenutzer sollen sie aber wie gewöhnliche Pushbuttons funktionieren.

»Ich konnte dieses gespaltene Verhalten realisieren, indem ich einfach die Unterklassen-Technik verwendet habe«, erklärt Geary. »Ich stelle meine eigene Fensterfunktion vor die Standardfunktion. In der Entwicklungsphase behandle ich viele Meldungen anders; zur Laufzeit lasse ich sie einfach durch.«

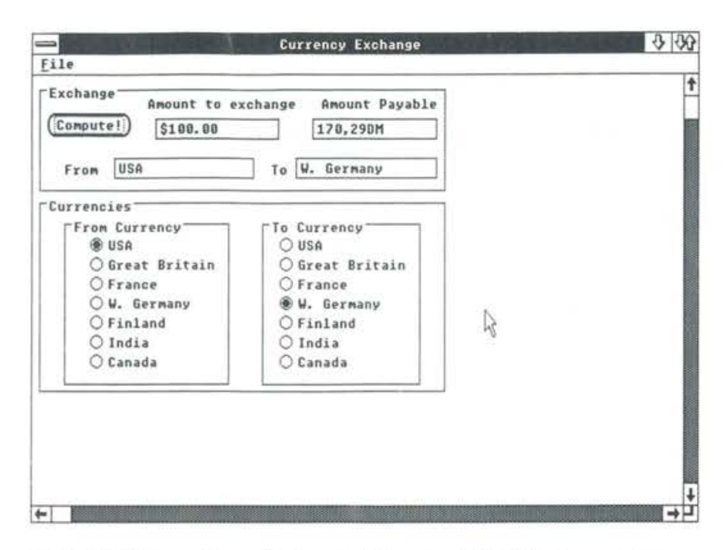


Bild 10: *Ein weiteres Beispiele für eine SQLWindows-Anwendung, die auch ohne Datenbankzugriffe auskommt: Die Umrechnung von Währungen.*

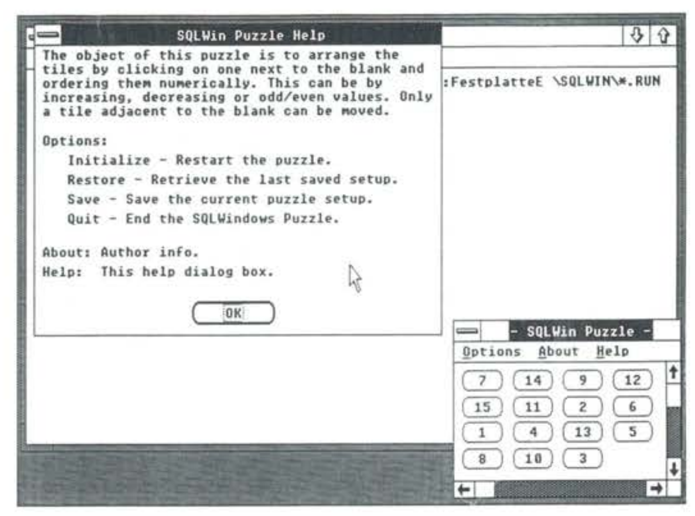


Bild 11: *Die Universaität einer Programmiersprache läßt sich sehr gut damit demonstrieren, daß man damit auch interaktive Spiele programmieren kann.*

Geary sagt, daß er eine Reihe von Fensterklassen von Grund auf neu erstellt hat, unter anderem ein Listenfenster, daß Spalten genauso behandeln kann, wie Zeilen, in etwa so wie in einer Tabellenkalkulation, und dabei noch Daten aus der Datenbank lesen kann (eine »normale« Windows-Anwendung erwartet, daß alle diese Daten im Speicher sind).

»Das schöne daran ist«, führt er aus, »daß meine eigenen Fensterklassen mit den eingebauten Windows-Klassen koexistieren und es gibt nicht viel Besonderes, was ich tun muß, um meine Objekte Seite an Seite mit den vorhandenen arbeiten zu lassen. Der Rest meines Programms behandelt sie alle gleich; es kümmert sich nicht darum, ob das Fenster von mir stammt oder von Windows.«

Ressourcenverwaltung

Andere Vorteile der Programmierung für Windows sind in der Speicherverwaltung und der Tastaturbehandlung zu sehen. Da jedes Programm unter Windows relokatier- und auslagerbar ist, ist der Programmierer laut Geary von Problemen wie der Speicherzerstückelung und dem Jonglieren mit Programmoverlays befreit.

»In Programmen, die ich in der Vergangenheit geschrieben habe,« so erklärt er, »mußte ich häufig Overlays verwenden. Und es ist immer schwierig zu entscheiden gewesen: 'Wie stark setze ich Overlays ein? Wie groß ist die Zielmaschine? Für wen optimiere ich?' Ich habe diesen Prozeß mehrere Male durchgemacht und es ist schrecklich. Man kann es nicht allen recht machen.«

»Doch in Windows braucht man ein Programm nur in eine Reihe von Codesegmenten aufzuteilen. Jedes Codesegment ist, was das Laden in den Speicher angeht, eine unabhängige Einheit und Windows lädt einfach so viele, wie passen. Wenn kein Platz mehr da ist, verschiebt es sie oder löscht sie aus dem Speicher, ganz wie es erforderlich ist. Mit anderen Worten, es optimiert für all die verschiedenen Systeme auf einmal.«

»Man muß etwas mehr Arbeit hineinstecken, um Vorteile aus der Speicherverwaltung von Windows zu ziehen, doch das ist sehr viel besser, als es mit einer starren Overlaystruktur zu tun zu haben.«

Genau wie bei der Speicherverwaltung begrüßt Geary die Tatsache, daß Windows den Programmierer mit Informationen versorgt, die die Notwendigkeit der direkten Hardwareprogrammierung überflüssig machen.

»In Windows«, sagt er, »ist die Tastaturbehandlung sehr einfach. Es werden einem Meldungen für alles geschickt. Mit jedem Tastendruck erhält man `WM_KEYDOWN`. Wenn eine Taste losgelassen wird, erhält man `WM_KEYUP`. Man kann zwischen der ersten Betätigung einer Taste und solchen, die durch den Auto-Repeat-Mechanismus der Tastatur entstanden sind unterscheiden.

»Im Grunde kann man die Tastatur auf jeder gewünschten Ebene behandeln, ohne ein 'unsauberes' Programm zu sein. Es gibt keinen Grund, direkt auf die Hardware zu gehen, denn es sind alle Dinge vorhanden, die man braucht, um das zu tun, was man tun will.«

Schließlich weist Geary noch auf die Vorteile hin, die, für Programmierer und Benutzer gleichermaßen, durch DDE und die Zwischenablage von Windows entstehen. Er glaubt, daß ein gut definierter Standard für die Kommunikation zwischen Anwendungen einen großen Einfluß darauf haben wird, welche Art von Software die Leute entwickeln werden.

»Philosophisch betrachtet ermöglichen die Zwischenablage und DDE den Leuten, davon wegzukommen, komplexe Alles-in-Einem-Programme zu bauen und sie öffnen den Weg dahin, kleinere Programme zu erstellen die effektiv zusammenarbeiten.«

Craig Stinson

Eine Systemutility für Windows:

Screen-Save-Funktion für MS-Windows

In diesem Artikel wird eine Applikation für MS-Windows entwickelt, die nach einer einstellbaren Zeit des »Nichtstuns« am Rechner (keine Tastatur- oder Mauseingabe) den Bildschirm dunkel schaltet, bis wieder eine Eingabeaktion stattfindet. Hierbei wurde durchgängig auf die Möglichkeiten von Windows 2.03 zurückgegriffen. Unter anderem wird der Systemzeitgeber verwendet und System-Eingriffe (sogenannte Hooks) vorgenommen.

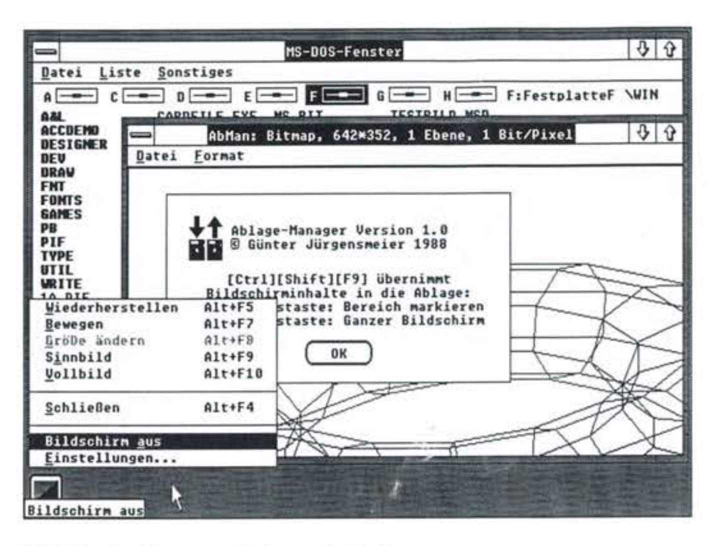


Bild 1: *ScrSave nach dem Aufruf*

Aufgabe einer Screen-Save-Funktion

Wird längere Zeit an einem eingeschalteten Rechner nicht gearbeitet, besteht die Gefahr, daß sich die ständig ungeänderte Bildschirmanzeige in den Bildschirm »hineinfrißt«, d.h. in der Phosphorschicht der Röhre häßliche Flecken zurückläßt. Dies gilt vor allem dann, wenn längere Zeit hell leuchtende Bereiche angezeigt werden. Da bei MS-Windows häufig »schwarz auf weiß« gearbeitet wird, sollte daher hier auf jeden Falle bei Eingabe-Pausen der Bildschirm abgeschaltet werden. Schön wäre es, wenn dies automatisch geschehen würde, d.h. wenn das System erkennt, daß längere Zeit nichts eingegeben wurde, es den Bildschirm ausschaltet. Sobald die Arbeit wieder aufgenommen wird (eine Taste gedrückt oder die Maus bewegt wird) soll der Bildschirm wieder einschaltet werden, wobei natürlich die alte Information wieder zu sehen sein soll. Diese Aufgabe wird von sogenannten Screen-Save-Programmen übernommen, die fast jeder EGA- oder VGA-Karte beigelegt werden.

Unter DOS arbeiten solche Programme nach folgendem Prinzip: Es wird der System-Tick (Interrupt 1CH, meldet sich jede 1/18 Sekunde) und der Keyboard-Interrupt (09H, meldet sich, wenn eine Taste gedrückt oder losgelassen wird) umgelenkt und ausgewertet. Wurden so viele System-Tick-Nachrichten gezählt, daß eine bestimmte Zeitspanne (zum Beispiel 5 Minuten) verstrichen ist und währenddessen keine Taste gedrückt worden ist, wird der Video-Signal-Ausgang abgeschaltet (Screen-Save), so daß der Bildschirm nun schwarz ist. Sobald wieder eine Taste gedrückt wird, wird der Ausgang sofort wieder eingeschaltet, so daß der alte Bildschirminhalt erneut sichtbar ist. Dieses Verfahren hat mehrere Nachteile:

- Da der Bildschirm während der Save-Phase komplett schwarz ist, läßt sich nicht mit einem Blick feststellen, ob nun der Monitor mit dem Netzschalter ausgeschaltet worden ist oder lediglich vom System abgeschaltet wurde. Man vermißt einen kleinen hellen Fleck oder ähnliches am sonst schwarzen Bildschirm, der eindeutig auf den gesicherten Bildschirm-Status hinweist.
- Weil lediglich der Tastatur-Interrupt als Aktivitätsmelder verwendet wird, wird bei einer intensiven Mausbenutzung (zum Beispiel in einem CAD-Programm) der Bildschirm ausgeschaltet, obwohl gearbeitet wird. Dies ist störend und führt leicht zu Fehleingaben.
- Das größte Problem ergibt sich jedoch bei Verwendung der erhältlichen Screen-Save-Funktionen unter Windows. Hier werden auch Tastendrücke nicht mehr als Aktivität erkannt, da das Windows-System den Tastatur-Interrupt umlenkt und sich nicht mehr darum kümmert, daß eine Screen-Save-Funktion auf diesen Interrupt wartet. So schaltet sich dann auch bei intensiver Tastatur-Benutzung in Windows nach der eingestellten Zeit der Bildschirm ab und läßt sich ohne weiteres nicht mehr reaktivieren (Tip: Drücken Sie in einem solchen Fall zur Reaktivierung des Bildschirms zum Beispiel die Scroll- oder die Shift-Lock-Taste).

Man kommt also nicht herum, für Windows eine neue Screen-Save-Funktion zu schreiben, die dann natürlich eine echte Windows-Applikation ist. Wie häufig beim Übergang von DOS zu Windows ergeben sich dabei völlig andere Lösungsansätze und auch völlig andere Probleme.

Lösung unter MS-Windows

Man kann sich die Sache einfach machen und innerhalb von Windows den Tastatur-Interrupt und den Maus-Eingabe-Interrupt (des seriellen Schnittstellentreiber COM1 oder COM2) wie bei der DOS-Funktion umlenken und entsprechend den Bildschirm abschalten. Doch würde eine solche Lösung einem wichtigen Windows-Prinzip widersprechen, der Hardware-Unabhängigkeit.

Diese wird dadurch erreicht, daß man als Aktivitätskontrolle alle Windows-Nachrichten (Messages) abfängt, die als Folge einer Eingabe an die Applikationen gesandt werden und statt der Video-Signalabschaltung den Bildschirm explizit schwarz zeichnet. Natürlich muß bei der

ACHTUNG

Wir sind ein junges, aufstrebendes Unternehmen und entwickeln und vertreiben leistungsfähige Software für den PC-Markt. Zu unserem Betätigungsfeld gehören unter anderem Textverarbeitung, Tabellenkalkulation und Datenbank. Unser Textverarbeitungsprogramm StarWriter PC gehört zu den meistverkauftesten in Deutschland, innerhalb von 1 1/2 Jahren wurden weit über 30000 StarWriter PC verkauft. Zur Entwicklung neuer Programme und Versionen suchen wir

Softwareautoren

Voraussetzung sind gute Kenntnisse des IBM PC und der Programmiersprache C. Das Aufgabenfeld wird die Weiter- und Neuentwicklung von Programmen für den IBM PC unter MS-DOS, MS-WINDOWS und dem MS-OS/2 Presentation Manager umfassen. Gearbeitet wird im Team, aber auch Einzelprojekte sind möglich. Voraussetzung ist eine Mindestarbeitsleistung von 30 Stunden pro Woche. Die Verdienstmöglichkeiten sind sehr gut und die Art der Zusammenarbeit kann auf flexible Weise vereinbart werden. Ein Wohnungsortwechsel ist in den meisten Fällen nicht erforderlich. Bei Interesse senden Sie bitte den Info-Coupon ein oder setzen sich direkt telefonisch mit Herrn Riemenschneider (04131 - 700912) in Verbindung.

IBM PC ist ein eingetragenes Warenzeichen der IBM Corp, und MS-DOS, MS-WINDOWS und MS-OS/2 Presentation Manager der Microsoft Inc.

☐ Ja, ich habe Interesse, bitte schicken Sie mir Informationen zum Thema „Softwareautor"



Name, Vorname

Straße, Nr.

PLZ, Ort

Bitte ausschneiden, auf eine Postkarte kleben oder in einen Briefumschlag stecken und an STAR-DIVISION GmbH, Postfach 2830 in 2120 Lüneburg abschicken.

StarDivision

Postfach 2830, 2120 Lüneburg, Telefon: (04131) 70090

Reaktivierung des Bildschirms dafür gesorgt werden, daß der alte Bildschirm wieder vollständig dargestellt wird. Naiv könnte dies dadurch geschehen, daß man den Bildschirm vor dem Schwarz-Zeichnen Punkt als Punktbild (Bildmap) in einen Puffer liest, den man bei der Reaktivierung wieder ausgibt. Das hat jedoch unter Windows einen Haken: Die Bildschirmdaten können unter Umständen sehr umfangreich sein (zum Beispiel 256 Kbyte). Soviel Speicher wird aber oft von Windows gar nicht zur Verfügung gestellt. Man müßte eine umfangreiche und komplizierte Datenauslagerung auf Platte vornehmen. Jede Bildschirm-Abschaltung und Reaktivierung würde viel Zeit in Anspruch nehmen.

Die Lösung ist zunächst viel einfacher, als Sie wahrscheinlich denken. Schließlich muß unter Windows jede Applikation ihr Bildschirmfenster selbst reaktivieren können. Dies wird zum Beispiel jedes Mal dann gemacht, wenn eine Applikation von der Vollbild-Darstellung auf die Sinnbild-Darstellung verkleinert wird. Man wird also einfach dadurch den Bildschirm schwarz zeichnen, indem man die Screen-Save-Applikation zum Vollbild vergrößert, dieses komplett schwarz malt und ihn wieder reaktivieren, indem die Applikation zum Sinnbild verkleinert wird. Die Neuzeichnung des Bildschirms erfolgt dann durch die einzelnen angezeigten Applikationen bzw. die Hintergrund-Applikation ohne daß man sich darum explizit kümmern müßte. So viel zur Windows-Theorie.

Wer seit längerer Zeit unter Windows programmiert, weiß, daß wegen dessen Komplexität Theorie und Praxis oft verschiedene Wege gehen. Man denkt sich eine tolle Sache aus, die, wenn man die Philosophie von Windows verstanden hat, eigentlich funktionieren müßte und stellt dann beim ersten Probelauf fest, daß alles ganz anders ist. Ursprünglich hatte ich vorgehabt, das Applikationsfenster so zu definieren, daß es als Vollbild keinen Rand besitzt und somit der (schwarze) Zeichen-Bereich (Client-Area) den gesamten Bildschirm ausfüllt. Doch so sehr man sich bei der Zusammenstellung der `CreateWindows`-Optionen auch abplagt: Man schafft es nicht, ein Applikationsfenster ohne Rahmen und Titelbalken (Title bar, Capion bar) zu erzeugen. Im zweiten Ansatz hatte ich vor, den Fenster-Rahmen (die sogenannte Non-Client-Area) explizit schwarz zu zeichnen, ein aufwendiges Unterfangen.

Beim Analysieren der vorhandenen Windows-Nachrichten ergab sich eine viel einfachere Lösung: Die Nachricht `WM_NCCALCSIZE` übergibt die Größe des Applikationsfenster einschließlich Rahmen (bei Vollbild-Darstellung also der gesamte Bildschirm). Normalerweise übergibt eine Windows-Applikation diese Nachricht unverändert an `DefWindowProc`. Diese verkleinert dann die Fenstergröße durch Abziehen der eingestellten Rand- und Capture-Breite auf die Client-Area-Größe. Wenn man nun einfach bei `WM_NCCALCSIZE` die Fenstergröße unverändert zurückgibt, nimmt die Client-Area das gesamte Applikationsfenster und damit bei Vollbild-Darstellung den gesamten Bildschirm ein. Soweit die Technik zum Abschalten des Bildschirms.



Bild 2: *ScrSave nach dem Sichern des Bildschirms*

Die Erkennung, ob eine Eingabe vorliegt, ist ebenfalls nicht ganz einfach. Schließlich gilt es ja nicht, die Eingabe-Nachricht für die Screen-Save-Applikation allein zu erkennen, sondern die Eingabe-Nachrichten für alle geladenen Applikationen. Hierzu bedient man sich eines Systemeingriffs, mit dem man alle gewünschten Nachrichten zum Analysieren umlenkt, bevor sie an die einzelnen Applikationen verteilt werden. Unter Windows wird hierzu die Funktion `SetWindowsHook` verwendet. Hier funktioniert offensichtlich einiges anders, als es im Windows-Referenz-Handbuch beschrieben wird. Doch besprechen wir zunächst den Rahmen der Applikation ScrSave.

Die Applikation ScrSave

Nach dem Aufrufen der Applikation ScrSave (einzudeutschen zum Beispiel als BildAus) verkleinert sie sich zu einem Sinnbild, dem einzigen konventionellen Fensterzustand, den sie annehmen kann. Zu Anfang erscheint jedoch eine Dialogbox (*Bild 1*), in der angegeben werden kann, ob die Funktion aktiv sein soll und die gewünschte Zeit eingetragen werden kann, nach wieviel Minuten »Nichtstun« der Bildschirm abgeschaltet wird. Die Standardwerte werden aus `WIN.INI` eingelesen. Nach dem Schließen der Applikation werden entsprechend die zuletzt eingestellten Werte in `WIN.INI` zurückgeschrieben. Ist die eingestellte Zeit verstrichen, wird der Bildschirm abgeschaltet (mit Schwarz gefüllt). Um zu erkennen, daß er dennoch physikalisch eingeschaltet ist, erscheint im unteren Bildschirmbereich die Sanduhr. Damit sich nun nicht diese an ihrer Abbildungsstelle in den Bildschirm »einfrißt«, wird sie langsam vom linken zum rechten Bildschirmrand und zurück bewegt. Sobald die Maus bewegt wird oder eine Taste gedrückt wird, wird der alte Bildschirminhalt sofort wieder angezeigt und der Cursor steht wieder auf der alten Eingabeposition, als wäre sonst nichts geschehen.

Die Quelldateien der Applikation

Bild 3 zeigt SCRSAVE.C, das C-Programm der Applikation. Da es sich um eine international anwendbare Applikation handelt, wurden alle landessprachenabhängigen Teile in die Resourcen-Datei SCRSAVE.RC (*Bild 4*) verlagert. Man kann daher die Applikation für andere Länder anpassen, ohne daß hierzu der Quellcode nötig wäre. Letzteres ist in Zukunft bei Programmentwicklungen für den europäischen Binnenmarkt sehr wichtig.

Zeichenketten aus der Resource-Datei werden mit LoadString in das C-Programm geladen. Die Definitionsdatei für den Linker, SCRSAVE.DEF, ist in *Bild 5* dargestellt. Alles wird mit der MAKE-Datei in *Bild 6* übersetzt und gelinkt. Vorher müssen Sie allerdings noch das Sinnbild der Applikation SCRSAVE.ICO eingeben, *Bild 9* zeigt es in seiner »Entwicklungsumgebung«, dem ICONEDIT.

Soweit vieles, was bereits von anderen Windows-Programmen bekannt ist. Im folgenden soll auf die Besonderheiten der Applikation eingegangen werden.

Das Hauptprogramm WinMain

Zum Hauptprogramm der Applikation gibt es nicht viel zu sagen: Zunächst wird der Initialisierungs-Teil der Applikation aufgerufen. Anschließend wird die bekannte Windows-Nachrichten-Bearbeitungsschleife aufgerufen, die bis zum Beenden der Applikation die Nachrichten an die Windows-Funktion fwMain weitergibt. Die Aufteilung in Hauptprogramm und Initialisierung sieht etwas künstlich und vielleicht auch umständlich aus. Beim Autor wird jedoch jede Windows-Applikation von einem festen Rahmen ausgehend entwickelt und die Initialisierung wurde vom Hauptprogramm getrennt, da bei größeren Applikationen der Initialisierungscode in einem getrennten Modul abgelegt wird, welches im Gegensatz zu WinMain und fwMain als entfernbar aus dem Speicher (DISCARDABLE) definiert wird. Die Auftrennung in Module wäre auch bei ScrSave sinnvoll, es wurde jedoch der Übersichtlichkeit wegen darauf verzichtet.

Initialisierung der Applikation

Jede Windows-Applikation unterscheidet zwischen zwei Arten der Initialisierung: Die Initialisierung des ersten Vorkommens der Applikation und die Initialisierung jedes weiteren aufgerufenen Vorkommens. Erstere wird in der Funktion InitWindow vorgenommen, die zweite in InitInstance. In InitWindow wird zunächst die Fensterklasse definiert. Man beachte die Hintergrundfarbe »schwarz« und den Verzicht auf einen vordefinierten Cursor. Anschließend werden benötigte Daten aus WIN.INI geladen. Dies erfolgt in der Funktion LoadWinIniData durch Verwendung der Windows-Funktionen GetProfileInt und GetProfileString.

```
/***************************************************************

 -------------- MS-Windows Application "SCRSAVE" --------------

================================================================

This MS-Windows application writes the complete screen black
after a delay without any mouse movement or key pressing and
reactivates it if a key is pressed or the mouse is moved.

================================================================

Copyright 1988 by
         Marcellus Buchheit, Buchheit software research
         Zaehringerstrasse 47, D-7500 Karlsruhe 1
         Phone (0) 721/37 67 76    (West Germany)

All rights reserved --- Release 1.00 of 88-Aug-01

***************************************************************/

#define NOMINMAX
#include <WINDOWS.H>
#include <stdlib.h>
#include <string.h>
#include "DEFS.H"

/***************************************************************

 ------------------ Application Variables -------------------

***************************************************************/

char  zAppName[] = "SCRSAVE";  /* Application name */
char  zAppTitle[40];           /* Application title */

/* Delay time in minutes betw. no action and screen saving. */

int  vSaveDelay;
int  vWIDelay; /* copy of WIN.INI value */

/* Flag: non-zero if screen saving mode is active */

BOOL bSaveActive;
BOOL bWIActive; /* copy of WIN.INI value */

/* Minutes Tick Count value for determing Screen Off */

int vSaveTick;

/* Flag: non-zero if screen is saved */

BOOL bSavedScreen;

/* Last input event cursor point */

POINT pEventCursor;

/* Screen size values */

int sxScreen;
int syScreen;
int sxCursor;
int syCursor;


/* ---------
   Instances
   ---------
*/

HANDLE hiMain;      /* Instance of application */


/* --------------
   Window Handles
   --------------
*/

HWND hwMain;      /* Main Window */
```

```
/* ----------------------------------
   Window & Dialog Procedure Pointers
   ----------------------------------
*/

FARPROC rfdMode;

/* ---------------------
   Event Hook Management
   ---------------------
*/

FARPROC rfhEvent;
FARPROC rfhPrevEvent;

/* ----------------------------
   Application defined Messages
   ----------------------------

   AM_ACTIVATE: sent if a screen must be repainted
   AM_SAVE:     sent if a screen must be saved
*/

#define AM_ACTIVATE WM_USER
#define AM_SAVE     (WM_USER+1)

/************************************************************

 --------------- Application specific functions -------------

************************************************************/

/************************************************************
 E r r o r N o M e m
=============================================================

The error message "Not enough memory" is displayed in a message
box. Normally, the error string is read as a string resource by
the index <STNOMEM>. But if no string exists, the message is
printed in English.

Parameters:
 hw  is the window handle and NULL if no window exists.

Return:
 None

************************************************************/

void ErrorNoMem(hw)
  HWND hw;

 {char zBuf[80];
  int s;

  s=LoadString(hiMain,STERROR,zBuf,sizeof(zBuf));
  if (s==0) strcpy(zBuf,"Not enough memory.");
  MessageBox(hwMain,zBuf,zAppTitle,MB_ICONEXCLAMATION|MB_OK);
 } /* ErrorNoMem() */

/************************************************************
 S e t M o d e
=============================================================

This functions sets the current screen mode.
Two modes are possible:
1) bSave=FALSE: The active-screen mode is set if <bActive> is
                TRUE. The timer sends a message every minute.
                The event hook analyses all events and each
                event resets the save-delay-counter. If
                <bActive> is FALSE, never timer nor event hook
                are set.
2) bSave=TRUE:  The screen-saved mode is set. The screen is
                painted black (except the walking hourglass
                cursor). The timer sends a message every cursor
                walking step. The event hook is set. Each event
                stops the screen-saved condition.
```

Werden die entsprechenden Einträge `delay` und `save` im Block `screen save` nicht gefunden, werden die Ersatzwerte »5« (Minuten) und »yes« verwendet. Die Originalwerte aus `WIN.INI` werden getrennt abgelegt (in `vWIDelay` und `bWIActive`), damit bei einer Beendigung der Applikation festgestellt werden kann, ob `WIN.INI` mit den eventuell neu gesetzten Werten aktualisiert werden muß.

Eine dritte Aufgabe von `InitWindow` besteht darin, verwendete Systemgrößen zu bestimmen. Bei ScrSave wird die Größe des Bildschirms und eines Sinnbildes benötigt. Solche Werte werden mit `GetSystemMetrics` ermittelt.

Die weitere Initialisierung wird in `InitInstance` vorgenommen. Hierbei ist eine kleine Besonderheit zu beachten: Es macht wenig Sinn, mehrere Vorkommen von ScrSave aufzurufen. Aus diesem Grund wird ein weiterer Aufruf von ScrSave abgeblockt, indem eine Hinweis-Box ausgegeben und die neue Vorkommen wieder beendet wird.

Nach dem Erzeugen des Prozedur-Vorkommens mit `ProcInstance` wird mit `CreateWindow` das Applikationsfenster angelegt. Kann das Fenster nicht angelegt werden und wird `NULL` zurückgegeben, ist höchstwahrscheinlich zu wenig Speicher vorhanden. In diesem Fall wird eine Fehler-Hinweis-Box in der Funktion `ErrorNoMem` ausgegeben und die Applikation beendet.

Als nächstes wird das System-Menü durch zwei Einträge `CMD_TURNOFF` (»Bildschirm aus«) und `CMD_MODE` (»Einstellungen...«) vergrößert. Der erste dient zum expliziten und sofortigen Abschalten des Bildschirms. Stattdessen kann auch das Applikations-Sinnbild doppelt angeklickt werden. Durch den zweiten Aufruf können die Applikationsparameter in einer Dialogbox geändert werden. Beide Einträge mußten ins System-Menü gelegt werden, da es nur aus dem Sinnbild heraus aufgerufen werden kann.

Nachdem alles definiert ist, wird das Applikationsfenster mit `ShowWindow` als Sinnbild angelegt. Nun muß noch, wie eingangs erwähnt, die Dialogbox mit den Einstellungen aufgerufen werden. Dies erfolgt jedoch nur, falls der Eingangsparameter `vCmdShow` nicht `SW_SHOWMINNOACTIVE` ist. Damit ist sichergestellt, daß die Dialogbox nur beim expliziten Aufruf der Applikation aus MS-DOS heraus angezeigt wird, nicht jedoch, wenn sie aus dem `LOAD`-Feld von `WIN.INI` beim Starten von Windows geladen wird (was der normale Aufruf von ScrSave sein dürfte).

Die Dialogbox-Funktion fdMode

Die Verwaltung der Nachrichten für die Mode-Dialogbox (»Einstellungen...«) ist im großen und ganzen Standard. Die eingestellten Werte werden beim Verlassen der Dialogbox analysiert. Der Eingabewert für die Verzögerung muß im Bereich 1 bis 99 liegen. Der obere Wert wird durch die Länge des Editierfelds garantiert. Bei 0 oder anderen fehlerhaften Eingaben wird ein Piepser ausgegeben, der Eingabefokus auf das fehlerhafte Feld gesetzt und das Verlassen der Box abgebrochen. Andernfalls werden die neuen

Werte in globalen Variablen abgelegt. Wurde statt »OK« das Feld »Beenden« (IDEXIT) gewählt, wird eine WM_DESTROY-Nachricht an das Hauptfenster geschickt. Damit wird die Applikation beendet.

In SCRSAVE.RC steht in der Definition der Dialalogbox noch eine Besonderheit, die auf einen Windows-Implementierungsfehler hinweist: Damit die Dialogbox an derselben Stelle wie das Menü der Applikation steht (oberhalb des Sinnbilds) wurde als Y-Position der Dialogbox ein negativer Wert eingegeben. Die Anzeige erfolgt auch wie gewünscht. Steht das Sinnbild jedoch am oberen Bildschirmrand, wird zwar das Systemmenü ordnungsgemäß nach unten verschoben, so daß es vollständig im Bildschirm steht, nicht aber die Dialogbox. Bei dieser ist nur noch der untere Rand sichtbar. Wenn man weder die OK- noch die Beenden-Taste drücken kann, hängt sich die Applikation gewissermaßen auf. Da eine Sinnbildposition am oberen Bildschirmrand aber sehr ungewöhnlich ist, wurde die ungewöhnliche Dialogbox-Position beibehalten.

Die Windows-Funktion fwMain

Wie bei fast allen Windows-Applikationen steht auch der Kern von ScrSave in der Windows-Funktion, beim Autor fwMain genannt. Hier werden alle vom System empfangenen Nachrichten verarbeitet, sofern sie für die Applikation von Interesse sind. Zusätzlich wurden noch zwei Nachrichten vereinbart, die aus verschiedenen »Ecken« des Programms an fwMain geschickt werden. Dort erfolgt dann zentral die Auswertung der Nachrichten. Die Kommunikation mit selbstdefinierten Nachrichten innerhalb einer Applikation verbessert erheblich die Strukturierung eines Programms, insbesondere dann, wenn eine Nachricht nicht sofort bearbeitet werden muß, sondern »hinten angestellt« werden kann (abgesandt mit PostMessage).

Beginnen wir zunächst mit der Erläuterung der Standard-Nachrichten. WM_CREATE speichert lediglich den Fensterbezug (Window-Handle) an eine globale Variable, damit in der gesamten Applikation problemlos darauf zugegriffen werden kann. WM_NCCALCSIZE ermittelt normalerweise aus der Fenstergröße die Größe des Zeichenbereichs (Client-Area). Wie oben erwähnt wird durch Rückgabe von 0 dieser auf die gesamte Fenstergröße ausgedehnt, allerdings nur, wenn tatsächlich der Bildschirm abgeschaltet werden soll und bSavedScreen TRUE ist. Bei WM_SYSCOMMAND werden alle Befehle aus dem Systemmenü verarbeitet. Jeder Versuch, das Fenster irgendwie zu öffnen oder die Wahl des Turn-Off-Befehls (»Abschalten«) führt zum Abschalten des Bildschirms. Dies erfolgt dadurch, indem die Nachricht AM_SAVE an fwMain selbst gesandt wird. Wird der Mode-Befehl (»Einstellungen...«) gewählt, muß vor dem Aufruf der Box unbedingt die Abschalt-Automatik abgeschaltet werden, sonst hängt sich die Applikation auf, nachdem der Bildschirm gesichert wurde. Auch dies stellt vermutlich einen Implementierungsfehler bei Windows dar.

```
Parameters:
 bSave     is the mode flag: FALSE for setting mode "active";
           TRUE for setting mode "saved".
 bActive   determines if the the active/save mode is valid or
           not. if <bActive> is zero, no timer or event hooks
           are set.

Return:
 None

*************************************************************/

void SetMode(bSave,bActive)
  BOOL bSave;
  BOOL bActive;

 {static BOOL bEventSet=FALSE;
  static WORD iCurrentTimer=0;
  WORD vDuration;

  if (iCurrentTimer!=0)
   {/* destroy timer */
    KillTimer(hwMain,1); iCurrentTimer=0;
   } /* if */
  if (bEventSet)
   {/* destroy current event hook */
    UnhookWindowsHook(WH_JOURNALRECORD,rfhEvent);
    bEventSet=FALSE;
   } /* if */
  /* mode not active => return */
  if (!bActive) return;
  if (bSave)
   {/* set timer for saved mode */
    /* per second one full cursor movement */
    vDuration=1000/sxCursor;
   }
  else
   {/* set timer for active screen mode */
    vDuration=60000; /* one minute */
   } /* if */
  /* Set timer with specified duration */
  iCurrentTimer=SetTimer(hwMain,1,vDuration,NULL);
  if (iCurrentTimer==0)
   {/* Timer not set(too many timers started): print message */
    char zBuf[60];
    LoadString(hiMain,STTIMEOVL,zBuf,sizeof(zBuf));
    MessageBox(hwMain,zBuf,zAppTitle,MB_ICONEXCLAMATION|MB_OK);
    return;
   } /* if */
  /* set event hook */
  rfhPrevEvent=SetWindowsHook(WH_JOURNALRECORD,rfhEvent);
  bEventSet=TRUE;
 } /* SetMode() */


/*************************************************************

 ------------------- Dialog Box & Hook Functions --------------

*************************************************************/


/*************************************************************
 f d M o d e
==============================================================

This function processes all messages for the "Mode" dialog box.

Parameters:
 see definition of fwMain
Return:
 none

*************************************************************/

BOOL FAR PASCAL fdMode(hw,uMsg,WP,LP)
  HWND hw;
  WORD uMsg;
  WORD WP;
  DWORD LP;
```

```
{BOOL b;

 switch (uMsg)
  {case WM_INITDIALOG:
     CheckDlgButton(hw,IDACTIVE,bSaveActive);
     SetDlgItemInt(hw,IDDELAY,vSaveDelay,FALSE);
     return TRUE;
   case WM_COMMAND:
     switch (WP)
      {case IDEXIT:
       case IDOK:
         /* check & save edit fields */
         bSaveActive=IsDlgButtonChecked(hw,IDACTIVE);
         vSaveDelay=GetDlgItemInt(hw,IDDELAY,&b,FALSE);
         if (!b||vSaveDelay==Ø)
          {/* illegal delay value: focus to error location */
           MessageBeep(NULL);
           SetFocus(GetDlgItem(hw,IDDELAY));
           /* mark full buffer */
           SendDlgItemMessage
            (hw,IDDELAY,EM_SETSEL,Ø,MAKELONG(Ø,32767));
           break; /* don't quit dialog box */
          } /* if */
         if (WP==IDEXIT)
          {/* send WM_DESTROY message to application window */
           PostMessage(hwMain,WM_DESTROY,Ø,ØL);
          } /* if */
         EndDialog(hw,FALSE);
         return TRUE;
      } /* switch */
  } /* switch */
 return FALSE;
} /* fdMode() */



/*************************************************************
 f h E v e n t
==============================================================

This function processes all event messages of the Windows
Systems. Each valid event resets during the "Active Screen
Phase" the screen save delay vounter and activates the screen
during the "Screen Save Phase".

Parameters:
 Code  contains the kind of message.
 WP    is the WORD Parameter.
 LP    is the DWORD Parameter.

Return:
 The event hook result is returned.

*************************************************************/

DWORD FAR PASCAL fhEvent(vCode,WP,rv)
 int vCode;
 WORD WP;
 WORD far *rv;

 {WORD uMsg;

  if (vCode<Ø)
   {/* Default action */
    return
     DefHookProc(vCode,WP,(LONG)rv,(FARPROC FAR*)
                           &rfhPrevEvent);
   } /* if */
  uMsg=*rv; /* message code */
  if (uMsg!=WM_MOUSEMOVE && uMsg!=WM_KEYDOWN &&
      uMsg!=WM_LBUTTONDOWN && uMsg!=WM_RBUTTONDOWN &&
      uMsg!=WM_MBUTTONDOWN)
   {/* no valid event message */
    return ØL;
   } /* if */
  if (uMsg==WM_MOUSEMOVE)
   {/* special condition "mouse moved" */
    RECT wrc;
    POINT pNewLoc;
```

Es spielt sich folgendes ab: Die Dialogbox ist ein Pop-Up-Menü, welches die Kontrolle zum Programm erst nach ihrer Beendigung (zum Beispiel durch Drücken von OK) zurückgibt. Wird nun einige Zeit in der Box nichts angezeigt, schaltet der Zeitgeber den Bildschirm ab. Dies erfolgt »hinter dem Rücken« der Dialogbox (und wurde wohl von den Windows-Vätern nicht eingeplant). Diese wird nicht schwarz überzeichnet. Aber bei der Reaktivierung des Bildschirms ist sie verschwunden. Da sie jedoch noch aktiv ist, kann kein Befehl mehr im Systemenü von ScrSave aufgerufen werden.

Auch beim Aufruf des Systemmenüs und Stehenlassen wird dieses nicht geschwärzt, sondern bleibt einsam im dunkeln Raum stehen. Dies ist zwar strenggenommen korrekt, sieht aber merkwürdig aus. Deshalb wird auch hier die Abschaltautomatik deaktiviert und beim Entfernen des Menüs wieder in den alten Zustand gebracht. Alles dies spielt sich in der Verarbeitung der Nachricht `WM_MENUSELECT` ab. Diese Nachricht wird jedesmal gesandt, wenn ein Feld im Menü gewählt wird. Nicht dokumentiert ist, wann das Menü wieder gelöscht wird, entweder weil ein Eintrag gewählt wurde oder weil [Esc] gedrückt bzw. die Maus außerhalb des Menüs positioniert wurde. Versuche haben ergeben, daß beim Abschalten jedesmal im oberen Wort des `DWORD`-Parameters der Nachricht 0 eingetragen wird. Sonst steht hier der Wert des Menü-Bezugs (Handle) und der ist nicht 0. Somit kann dieser Parameter als Identifizierung dienen. Bleibt zu hoffen, daß dies in der nächsten Windows-Version dokumentiert ist und nicht plötzlich geändert wird.

Das Abschalten des Bildschirms

Die Nachricht `AM_SAVE` erscheint, wenn der Bildschirm abgeschaltet werden soll. Es müssen einige Systemwerte gerettet werden. Hierzu zählt das Sichern des Eingabefokus, denn die Vergrößerung des Fensters ScrSave auf Bildschirmgröße weist der Applikation den Fokus zu. Doch nach dem Reaktivierung des Bildschirms möchte man ja an der alten Stelle weiterarbeiten.

Nach dem Sichern des Fokus wird das Fenster vergrößert und schwarz gezeichnet. Die Cursor-Position und das Cursor-Erscheinungsbild muß ebenfalls gerettet werden. Nun wird die Startposition des Cursors berechnet, der während der Bildschirmabschaltung am Bildschirm hin- und herfährt. Der Cursor wird als Sanduhr (`IDC_WAIT`) definiert und auf die berechnete Position gesetzt. Wenn Sie sich wundern, warum der Cursor mit `ShowCursor` noch einmal eingeschaltet wird, obwohl er doch schon sichtbar ist: Dies ist ein Spezialfall für jene zu bemitleidenden Leute, die Windows ohne Maus bedienen müssen. Wenigstens in der Ausschaltzeit des Bildschirms bekommen sie jetzt einen Cursor zu sehen.

Schließlich wird der Applikationsstatus in `SetMode` von »eingeschaltet« auf »aufgeschaltet« geändert (siehe unten).

Insbesondere wird der Zeitgeber so umdefiniert, daß er sich nicht mehr jede Minute meldet, sondern so häufig, daß die Sanduhr in einer kontinuierlichen Bewegung hin- und herfährt.

Das Reaktivieren des Bildschirms

Das Gegenteil der Nachricht `AM_SAVE` ist `AM_ACTIVATE`. Sie wird an `fwMain` gesandt, wenn der Bildschirm reaktiviert werden soll. Diese Nachricht stammt ausschließlich von der Eingriffsfunktion `fhEvent` (siehe unten). Das Reaktivieren des Bildschirms ist einfach: Das Applikationsfenster wird wieder auf Sinnbildgröße verkleinert. Dadurch zeichnet sich der Bildschirm wieder auf den alten Stand. Der Cursor wird wieder auf die alte Position in seiner alten Form gesetzt und für Nicht-Maus-Benutzer wieder abgeschaltet. Ebenso wird der Eingabefokus dem Fenster zugewiesen, das es vor dem Bildschirm-Abschalten einnahm. Es hat sich also nichts geändert, außer daß der Bildschirm geschont wurde.

Die Nachricht vom Zeitgeber

Jede Nachricht von dem gesetzten Zeitgeber wird als `WM_TIMER` an die Windows-Funktion gesandt. Hier werden zwei unterschiedliche Aktivitäten ausgelöst, je nachdem ob der Bildschirm aktiv oder abgeschaltet ist: Im ersteren Fall wird jede Minute der Verzögerungszähler erhöht. Überschreitet der Wert den eingestellten Wert (in `vSaveDelay`), so war keine Eingabe in dieser Zeit erfolgt und die Abschaltautomatik sendet eine `AM_SAVE`-Nachricht ab, die den Bildschirm abschaltet (siehe oben).

Die Verarbeitung der Zeitgeber-Nachricht beim abgeschalteten Bildschirm ist etwas komplizierter: Hier muß das Cursor-Sinnbild eine Position weiterbewegt werden. Dies darf jedoch nur solange erfolgen, wie nicht der linke bzw. rechte Bildschirmrand erreicht ist. Dann muß die Bewegungsrichtung umgekehrt werden. `SetCursorPos` setzt die neue Cursor-Position.

Beenden der Applikation

`WM_DESTROY` bezeichnet bekanntlich den Wunsch, die Applikation zu beenden. Neben der Aktualisierung der `WIN.INI`-Datei muß hier vor allem der Systemeingriff entfernt werden, sonst stürzt nach dem Entfernen des Applikations-Codes aus dem Speicher das Windows-System sofort ab.

Eine erwähnenswerte Originalität besitzt ScrSave noch: Sie dürfte vermutlich eine der wenigen sinnvollen Applikationen darstellen, die ohne Bearbeitung der `WM_PAINT`-Nachricht auskommen. Das Schwarz-Zeichnen des Bildschirms wird durch Löschen des Fensterhintergrunds ausgelöst da dieser laut Definition der Fensterklasse schwarz ist.

```
    SetRect
     (&wrc,pEventCursor.x-sxCursor,pEventCursor.y-syCursor,
           pEventCursor.x+sxCursor,pEventCursor.y+syCursor
     );
    GetCursorPos(&pNewLoc);
    /* update cursor point */
    memcpy((char*)&pEventCursor,(char*)&pNewLoc,sizeof(POINT));
    /* new mouse location within cursor rectange =>
       no activation */
    if (PtInRect(&wrc,pNewLoc)) return 0L;
   } /* if */
  if (bSavedScreen)
   {/* Screen is saved: activate it again */
   PostMessage(hwMain,AM_ACTIVATE,0,0L);
   }
  else
   {/* screen is active: reset delay counter */
   vSaveTick=0;
   } /* if */
 } /* fhEvent() */


/**************************************************************

 -------------------- Main Window function ------------------

**************************************************************/

LONG FAR PASCAL fwMain(hw,uMsg,WP,LP)
  HWND  hw;
  WORD  uMsg;
  WORD  WP;
  DWORD LP;

 {static HCURSOR hcuActive; /* cursor shape of active screen */
  static POINT pActCursor;  /* cursor point/active screen */
  static POINT pSaveCursor; /* cursor point/saved screen */
  static int vCursorDrct;   /* cursor direction/saved screen */
  static hwFocus;           /* input focus before saving */
  static bMenuSave=FALSE;   /* bSaveActive-value during menu */
  int f;                    /* flag variable */

  switch (uMsg)
   {case WM_CREATE:
      /* set application window handle */
      hwMain=hw;
      return 0L;
    case WM_NCCALCSIZE:
      if (bSavedScreen)
       {/* set client area to full window area */
        return 0L; /* no change of rectangle at <LP> */
       } /* if */
      break; /* else: default activity */
    case WM_SYSCOMMAND:
      switch (WP)
       {case CMD_MODE:
          if (bSaveActive)
           {/* cancel screen save mode during dialog box */
            SetMode(FALSE,FALSE);
           } /* if */
          f=DialogBox
             (hiMain,MAKEINTRESOURCE(DB_MODE),hw,rfdMode);
          if (f==-1) ErrorNoMem();
          SetMode(FALSE,bSaveActive);
          return 0L;
        case SC_RESTORE:
        case SC_MINIMIZE:
        case SC_MAXIMIZE:
        case CMD_TURNOFF:
          if (!bSavedScreen)
           {/* send message for saving screen */
            SendMessage(hwMain,AM_SAVE,0,0L);
           } /* if */
          return 0L; /* message consumed */
       } /* switch */
      /* other sys menu commands: default action */
      break;
    case WM_MENUSELECT:
      /* application menu selection: stop save-screen during
         menu active */
```

```
    if (HIWORD(LP)!=Ø)
     {/* begin/during menu selection */
      if (bSaveActive)
       {/* save value & cancel screen save mode */
        bMenuSave=TRUE; bSaveActive=FALSE;
        SetMode(FALSE,FALSE);
       } /* if */
     }
    else
     {/* end of menu selection */
      bSaveActive=bMenuSave; bMenuSave=FALSE;
      if (bSaveActive) SetMode(FALSE,TRUE);
     } /* if */
    break; /* continue default action */
  case AM_SAVE:
    if (!bSaveActive) return ØL;
    /* Save Screen: set black window to maximize */
    hwFocus=GetFocus(); /* save current input focus */
    bSavedScreen=TRUE;
    ShowWindow(hwMain,SW_SHOWMAXIMIZED);
    /* set saved screen cursor "hourglass",
       save active screen cursor */
    hcuActive=SetCursor(LoadCursor(NULL,IDC_WAIT));
    GetCursorPos((LPPOINT)&pActCursor);
    pSaveCursor.x=sxCursor/2+1;
    pSaveCursor.y=3*syScreen/4;
    vCursorDrct=1; /* move from left to right */
    memcpy ((char*)&pEventCursor, (char*)&pSaveCursor,
                                  sizeof(POINT));
    SetCursorPos(pSaveCursor.x,pSaveCursor.y);
    /* display cursor for non-mouse system */
    ShowCursor(TRUE);
    SetMode(TRUE,TRUE); /* set saved-screen mode */
    return ØL;
  case AM_ACTIVATE:
    /* activate saved screen */
    if (bSavedScreen)
     {/* switched off and any user command:
         set screen save iconic */
      bSavedScreen=FALSE; /* reset flag */
      SetMode(FALSE,TRUE); /* active screen mode */
      ShowWindow(hwMain,SW_SHOWMINNOACTIVE);
      /* hide cursor for non-mouse system */
      ShowCursor(FALSE);
      /* set active screen cursor */
      SetCursor(hcuActive);
      SetCursorPos(pActCursor.x,pActCursor.y);
      SetFocus(hwFocus); /* set active screen input focus */
      return ØL; /* no further activity */
     } /* if */
    return ØL;
  case WM_TIMER:
    /* analyse second/minute tick */
    if (!bSavedScreen)
     {/* minutes tick during "Screen Active Phase */
      vSaveTick++; /* increase tick counter */
      if (vSaveTick>=vSaveDelay)
       {/* turn off screen: Send Message to WPMain */
        PostMessage(hw,AM_SAVE,Ø,ØL);
        vSaveTick=Ø; /* reset counter */
       } /* if */
     }
    else
     {/* tick during "Screen Save Phase for moving cursor */
      pSaveCursor.x+=vCursorDrct;
      if (pSaveCursor.x<sxCursor/2 ||
          pSaveCursor.x>=sxScreen-sxCursor/2
         )
       {/* change direction of cursor moving */
        vCursorDrct=-vCursorDrct;
        /* set new location in inverse direction */
        pSaveCursor.x+=2*vCursorDrct;
       } /* if */
      /* set new cursor location */
      SetCursorPos(pSaveCursor.x,pSaveCursor.y);
     } /* if */
    return ØL;
  case WM_DESTROY:
  case WM_ENDSESSION:
    if (bSaveActive!=bWIActive)
```

Aktualisierung von WIN.INI

Beim Beenden der Applikation durch die Nachricht `WM_DESTROY` oder beim Beenden der gesamten Windows-Sitzung (mit der Nachricht `WM_ENDSESSION`) wird überprüft, ob die aus `WIN.INI` geladenen Daten in der Dialogbox geändert wurden. Falls ja, werden sie mit der Funktion `WriteProfileString` aktualisiert. Falls vorher noch kein Eintrag `Screen Save` in `WIN.INI` stand, wird er hiermit angelegt. Wie in *[3]* wird statt `sprintf` die Funktion `itoa` verwendet, um die Codelänge der Applikation zu verkürzen.

Systemeingriff mit SetWindowsHook

Neben der Bildschirmabschaltung stellt die Erkennung der Eingabe-Aktivitäten in der Applikation eine Schlüsselfunktion dar, die sich nicht ohne weiteres lösen ließ. Die Eingabe-Nachrichten an die Applikation ScrSave sind wertlos, da normalerweise mit anderen Applikationen gearbeitet wird und deren Eingabe-Nachrichten nicht an ScrSave gesandt werden. Als Lösung kommt nur ein Systemeingriff (Hook) in Frage. Ein Systemeingriff ist die Definition und der Einbau einer Funktion, die alle Nachrichten im Windows-Kern abfragt, sich die interessanten herausfiltert und analysiert und abschließend alle Nachrichten wieder an den Windows-Kern zurückgibt. *Bild 9* zeigt dies beispielhaft für die für ScrSave interessanten Eingabe-Nachrichten. Es gibt in Windows 2.03 eine Vielzahl von Eingriffen für die verschiedensten Zwecke. Sie werden mit der Funktion `SetWindowsHook` angelegt und mit `UnhookWindowsHook` wieder entfernt. Ein Parameter beschreibt die Art des Systemeingriffs.

`WH_KEYBOARD` ist zum Beispiel ein Wert für diesen Parameter. Die Eingriffsfunktion wird bei jeder gedrückten Taste aufgerufen. Doch die Mausbewegungen werden nicht erfaßt. Somit wäre dieser Eingriff nur beschränkt geeignet. Glücklicherweise gibt es den Eingriff `WH_JOURNALRECORD`, der für die Aufgabe hervorragend geeignet ist, obwohl er eigentlich für etwas völlig anderes dient: In manchen Applikationen (zum Beispiel Tabellenkalkulationen) müssen immer wieder dieselben Tastendrücke und Mausbewegungen durchgeführt werden, um eine bestimmte Aktion auszuführen. Die Arbeit kann dadurch erleichtert werden, daß einmalig alle Mausbewegungen und Tastendrücke bei der Ausführung der Aktion aufgezeichnet und in eine Art Makro gespeichert werden. Bei einem erneuten Ablauf der Aktion wird dann einfach statt den Bewegungen und Tastendrücken das Makro gestartet, welches die Eingabe selbständig simuliert. Zur Aufzeichnung der Bewegungen und Tastendrücke ist `WH_JOURNALRECORD` gedacht, zum erneuten Ablaufen `WH_JOURNALPLAYBACK`. Für ScrSave wird also `WH_JOURNALRECORD` zweckentfremdet, dies stört jedoch die Funktionsweise des Eingriffs in keinerlei Weise.

Setzen des Applikations-Zustands in SetMode

In der Funktion `SetMode` wird die automatische Bildschirmabschaltung und die Bildschirmabschaltung jeweils ein- und ausgeschaltet. Zwei Aktionen sind damit verbunden: Defininieren/Entfernen eines Systemeingriffs und Starten/Stoppen eines Zeitgebers mit der geeigneten Zeit.

Mit `SetWindowsHook` wird die Eingriffsfunktion `fhEvent` in das Windows-System eingefügt. Ab sofort werden alle Eingabe-Nachrichten an diese Funktion gesandt.

Hier geht es wieder leicht experimentell zu, jedenfalls sieht die Funktion leicht anders aus als im Windows-Reference-Manual beschrieben. Der dritte Parameter sollte laut diesem Handbuch auf eine Nachrichten-Struktur (`MSG`) zeigen, dies war jedoch definitiv nicht der Fall. Sie zeigt vielmehr auf einen einsamen Nachrichten-Wert ohne Parameter oder Zielfenster-Bezug. Ob sich hier die Handbuchautoren oder die Implementierer geirrt haben, bleibt unklar. Laut Reference-Manual sollte die Funktion überhaupt keinen Wert (`void`) zurückliefern, dies kollidiert jedoch mit der Vereinbarung der Funktion `DefHookProc`, die einen `DWORD`-Wert zurückliefern muß, wenn nicht das gesamte Windows-System abstürzen soll. Das Reference-Manual muß hier vermutlich wie folgt interpretiert werden: Grundsätzlich liefert jede Eingriffsfunktion (Hook-Function) einen `DWORD`-Wert zurück. Oft (und so bei `WH_JOURNALRECORD`), wird dieser Wert jedoch nur von `DefHookProc` verwendet.

Wie arbeitet nun die Eingriffsfunktion `fhEvent`? Wenn der Parameter `vCode` negativ ist, muß entsprechend der Regeln des Reference-Manuals `DefHookProc` aufgerufen werden und ihr Rückgabewert zurückgegeben werden. Diese Funktion wird zum Beispiel dann aufgerufen, wenn die Liste der Eingriffsfunktionen umgebaut wird. Deshalb besitzt sie auch einen Zeiger auf die Adresse der Nachfolger-Eingriffsfunktion (in `rfhPrefEvent`). Ist `vCode` nicht negativ, erfolgt die eigentliche Verarbeitung der Eingabe-Nachricht.

Handelt es sich hierbei um keine entscheidende Eingabe wie Taste heruntergedrückt oder Maus bewegt, spielt diese Nachricht für ScrSave keine Rolle, und die Eingriffsfunktion wird beendet. Sonst muß zwischen Tastendruck und reiner Mausbewegung unterschieden werden. Bei einem Tastendruck (Tastatur oder Maus) wird der Bildschirm immer reaktiviert. Bei einer Mausbewegung dagegen genügt es nicht, `WM_MOUSEMOVE` allein zur Reaktivierung zu verwenden. Der Grund liegt darin, daß `WM_MOUSEMOVE` nach bestimmten Systemaufrufen vom Windows-System gesendet wird, auch wenn die Maus physikalisch gar nicht bewegt wurde. Nachdem dies durch eine ziemlich aufwendige und undurchsichtige Flag-Verwaltung eliminiert worden war, trat ein anderes Problem auf: die Empfindlichkeit der Maus. Man brauchte nur mit der Hand auf den Tisch zu klopfen oder laut zu husten und schon hat eine empfindliche Logitech-Maus den Bildschirm wieder eingeschaltet!

```
       {/* store modified flag into WIN.INI */
        WriteProfileString
         ("Screen Save","Save",bSaveActive? "Yes":"No");
       } /* if */
      if (vSaveDelay!=vWIDelay)
       {char zBuf[10];
        /* store modified value into WIN.INI */
        WriteProfileString
         ("Screen Save","Delay",
          (LPSTR)itoa(vSaveDelay,zBuf,10)
         );
       } /* if */
      /* destroy timers & event hook */
      bSaveActive=FALSE; SetMode(FALSE,FALSE);
      if (uMsg==WM_DESTROY)
       {/* terminate application */
        PostQuitMessage(0); return 0L;
       } /* if */
      break;
  } /* switch */
 return DefWindowProc(hw,uMsg,WP,LP);
} /* fwMain() */


/**************************************************************

 --------- General & Windows dependent initialization ---------

**************************************************************/

/**************************************************************
 L o a d W i n I n i D a t a
==============================================================

Load all values from the file WIN.INI into the data areas which
are needed by the application.
This function is called only by the first instance of a window.
Later instances copy the data from the first instance by the
API function GetInstanceData() in the function PreviousData().

Parameters:
 none

Return:
 -1  if any error occures during reading WIN.INI file.
  0  if read correctly but no valid changes made.
  1  if read correctly and valid changes made.

**************************************************************/

int LoadWinIniData()

 {int v;
  char s[4];
  int f;

  f=0;
  /* Get delay value (in minutes) */
  v=GetProfileInt("screen save","delay",5);
  f|=vSaveDelay!=v; vWIDelay=vSaveDelay=v;
  GetProfileString("screen save","save","YES",s,sizeof(s));
  /* result: start with 'Y' => TRUE, otherwise => FALSE */
  v=strlen(s)>0 && (char)AnsiUpper(MAKEINTRESOURCE(s[0]))=='Y';
  f|=bSaveActive!=v; bWIActive=bSaveActive=v;
  return f; /* 0 or 1, no error possible */
 } /* LoadWinIniData() */

/**************************************************************
 I n i t W i n d o w
==============================================================

The application window is initialized. This function is called
only once. The created data are used for all instances of the
window.

Parameters:
 hiMain  is the handle to the (first) instance.

Return:
 TRUE is returned if initialization correct, otherwise FALSE
```



„Sind Sie Tallyaner?“

Weltweit gibt es immer mehr Tallyaner. Kein Wunder. Denn bei der Breite unseres Programms bieten wir jedem den richtigen Drucker. Zum Einsteigen, Aufsteigen und Umsteigen. Passend zu fast jedem System.
Wir bauen Drucker, die von sich reden machen, weil man so wenig dB (A) von ihnen hört, und solche, mit denen man es bunt treiben kann – in 12 Farben.
Sie haben die Wahl zwischen Druckern, die besonders schön, besonders schnell und besonders schnell schön drucken. Und wenn Sie häufig die zu verarbeitenden Papiere wechseln müssen – wir bauen sogar Drucker, die sich vollautomatisch umstellen. Um nur einige Vorteile zu nennen.

Werden Sie doch auch Tallyaner.

Schnelligkeit und Schrift oho – Tallyssimo!

Orgatechnik in Köln
vom 20. – 25. 10. 1988
Halle 4.1, Gang A9/B10

mannesmann *technologie*

Mannesmann Tally GmbH
Postfach 29 69, D-7900 Ulm

☐ Schicken Sie mir Ihr Händlerverzeichnis und Ihre Typenübersicht über Nadel-, Tintenstrahl-, Laser- und Hammerbankdrucker.

Name ____________________

Firma ____________________

Straße ____________________

PLZ ______ Ort ______________

Telefon ____________________

MWG 3866-200

```
*****************************************************************/

BOOL InitWindow()

 {WNDCLASS WndClass;

  /* create window class */
  WndClass.lpszClassName=zAppName;
  WndClass.hInstance=hiMain;
  WndClass.lpfnWndProc=fwMain;
  WndClass.hCursor=NULL;
  WndClass.hbrBackground=GetStockObject(BLACK_BRUSH);
  WndClass.style=0;
  WndClass.hIcon=LoadIcon(hiMain,zAppName);
  WndClass.lpszMenuName=NULL;
  WndClass.cbClsExtra=0; WndClass.cbWndExtra=0;
  /* register window, return if error */
  if (!RegisterClass(&WndClass)) return FALSE;

  /* Load data from WIN.INI */
  LoadWinIniData();

  /* Get System Parameters */
  sxScreen=GetSystemMetrics(SM_CXSCREEN);
  syScreen=GetSystemMetrics(SM_CYSCREEN);
  sxCursor=GetSystemMetrics(SM_CXCURSOR);
  syCursor=GetSystemMetrics(SM_CYCURSOR);

  /* return without error */
  return TRUE;
 } /* InitWindow() */

/*****************************************************************
 I n i t I n s t a n c e
==================================================================

The complete initialization of a new instance of the
application window. If no previous instance exists, the
application window is initialized and the common data for all
instances are created. Otherwise, common data are copied from
the previous instance to the new instance.
All individual data in the instance data segment are
initialized.
If memory is too small, a error message box is displayed and
FALSE is returned.

Parameters:
 hiPrev    is a handle to the first instance of the window.
 hiNew     is a handle to the new instance of the window.
 zCmdLine  points to the command line buffer.
 vCmdShow  is the entry style of the window.

Return:
 TRUE is returned if initialization complete,
 otherwise FALSE (memory too small).

*****************************************************************/

BOOL InitInstance(hiNew,hiPrev,zCmdLine,vCmdShow)
  HANDLE hiNew,hiPrev;
  LPSTR  zCmdLine;
  int    vCmdShow;

 {char zBuf[80];
  HMENU hmnSystem;
  int v;

  hiMain=hiNew; /* set instance global */
  LoadString(hiMain,STTITLE,zAppTitle,sizeof(zAppTitle));
  if (!hiPrev)
   {/* Initialize window, exit if error */
    if (!InitWindow()) goto MemError;
   }
  else
   {/* Application will be called only once */
    LoadString(hiMain,ST2NDINST,zBuf,sizeof(zBuf));
    MessageBox(NULL,zBuf,zAppTitle,MB_ICONASTERISK|MB_OK);
    return FALSE; /* return with error */
   } /* if */
```

Um dieses Problem zu lösen, wird deshalb in ScrSave geprüft, ob die Maus tatsächlich bewegt wurde. Dies wird dadurch festgestellt, indem die Position bei `WM_MOUSEMOVE` mit der zuletzt gesetzten verglichen wird. Liegt die neue Position außerhalb eines Feldes um die alte, das der doppelten Größe des Cursor-Sinnbildes entspricht, wird angenommen, daß die Maus tatsächlich bewegt wurde und die Eingabeaktion ist gültig. Andernfalls wird die Mausbewegung ignoriert. Durch diesen Trick wurde auch erreicht, daß die Maus vom Tisch angehoben und wieder an anderer Stelle abgelegt werden, ohne daß sich der Bildschirm wieder einschaltet. Auch konnte die oben erwähnte aufwendige Flag-Verwaltung wieder entfernt werden (Ich habe fünf Stunden gebraucht bis diese einwandfrei lief...).

Liegt eine gültige Eingabe-Nachricht vor und ist der Bildschirm gerade abgeschaltet (`bSavedScreen` ist `TRUE`), wird eine `AM_ACTIVATE`-Nachricht an `fhMain` gesandt. Die weitere Verarbeitung der `AM_ACTIVATE`-Nachricht wurde bereits oben beschrieben. Ist der Bildschirm aktiv, setzt jede Eingabe-Nachricht in `fhEvent` den Zähler, der die verstrichene Zeit bis zum Bildschirm-Abschaltzeitpunkt hochzählt, auf Null zurück. Somit wird das Abschalten bei jeder Eingabe erneut verzögert.

Besonderheiten beim Arbeiten mit Systemeingriffen

Das Programmieren mit Systemeingriffen verlangt einige tiefgehendere Kenntnisse des Windows-Systems. So hat das SS-Register innerhalb einer Eingriffsfunktion einen anderen Wert als das DS-Register. Somit muß die Windows-Applikation unbedingt mit der Compiler-Option `-Aw` übersetzt werden, damit der Compiler weiß, daß er DS und SS nicht einfach als synonym annehmen kann. Daß diese Annahme hierbei außerhalb der Eingriffsfunktion zu Compiler-Warnungen »SS!=DS« beim Aufruf der Zeichenketten-Bibliothek führt, kann ignoriert werden, da dort SS=DS vom Betriebssystem garantiert ist.

Vielleicht ist Ihnen schon aufgefallen, daß in `SCRSAVE.DEF` (*Bild 5*) das Code- und Data-Segment nicht `MOVEABLE` oder `DISCARDABLE` ist, sondern als `FIXED` vereinbart worden ist, also weder im Speicher verschoben noch während seiner Ausführung aus dem Speicher entfernt werden kann. Dies ist erforderlich, weil `WH_JOURNALRECORD` einen Kerneingriff von Windows darstellt (im Gegensatz zu `WH_MSGFILTER`) und der Code der Eingriffsfunktion ständig verfügbar sein muß. Ist der Code nun nicht `fixed`, kann er in einer EMS-(expanded memory system)-Umgebung in eine Speicherbank ausgelagert werden, die vom Kern aus nicht zugreifbar ist. Dies führt früher oder später zu einem Systemabsturz. In [1] wird deshalb empfohlen, den Code und die Daten der Eingriffsfunktion und damit die der ganzen Applikation als `fixed` zu vereinbaren. Er wird dann von der Speicherverwaltung nicht in den expanded memory ausgelagert. Im Gegensatz hierzu empfiehlt das Windows-

Reference-Manual, die Funktion in eine dynamische Link-Library zu verlegen, eine kompliziertere aber vielleicht effizientere Methode.

Setzen der Zeitgeber

In `SetMode` wird auch der Zeitgeber für die Ermittelung der Abschaltverzögerung und für die Bewegung des Sanduhr-Cursors während der Ausschaltphase gesetzt und gelöscht. Auf Zeitgeber wird in *[3]* detailliert eingegangen, so daß sich der Autor hier kurz fassen möchte. Der Zeitgeber für die Abschaltverzögerung meldet sich etwa jede Minute, so daß er kaum das System belastet. Während der Ausschaltphase wird er so umdefiniert, daß er den Takt für die Cursorbewegung vorgibt. Es wird ein Wert verwendet, bei dem das Sinnbild des Cursors innerhalb einer Sekunde seine Breite als Weg zurücklegt. Der Zeitgeber ist in diesem Fall sehr aktiv.

Bekanntlich ist der Zeitgeber nicht sonderlich genau. Da jedoch kaum jemand seine Uhr nach der Ausschaltverzögerung des Bildschirms stellen wird, ist dies hier nicht sonderlich wichtig.

Ausblick

Wie man sieht, kann man unter Windows auch unkonventionelle Probleme elegant lösen, obwohl es hier oder da noch ein wenig »klemmt«. Die Universalität von MS-Windows ist dem Weitblick der Windows-Designer bei Microsoft zu verdanken. Gleichzeitig wird auch der weitverbreitete Irrglaube widerlegt, Programmieren unter Windows würde wegen der Gleichartigkeit von Ein- und Ausgabe zu einer langweiligen und unkreativen Arbeit führen. Das Gegenteil ist der Fall, allerdings auf einem höheren Niveau: Der Programmierer wird seine Erfahrung und Kreativität nicht mehr in Ein-/Ausgabetreiber stecken, deren Entwicklung zeitraubend und frustrierend ist, sondern kann sich voll auf die mit seiner Applikation zu realisierenden Algorithmen konzentrieren, wobei er geschickt unter einer Vielzahl von Hilfsmittel (Windows-Funktionen und -Nachrichten) auswählen muß, um die Realisierung möglichst elegant erscheinen zu lassen. Was bietet mehr Abwechselung und Kreativität? Bleibt abschließend noch zu hoffen, daß die Applikation ScrSave ohne große neue Klimmzüge auch unter dem OS/2-Presentation-Manager läuft.

Marcellus Buchheit

```
 /* Create procedure instance addresses */
 rfdMode=MakeProcInstance(fdMode,hiMain);
 rfhEvent=MakeProcInstance((FARPROC)fhEvent,hiMain);

 /* Create Application's Window, branch if error */
 if (!CreateWindow
       (zAppName,zAppTitle,WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,
        CW_USEDEFAULT,NULL,NULL,hiMain,NULL
       )
    )
   goto MemError;

 /* Change system menu */
 hmnSystem=GetSystemMenu(hwMain,FALSE);
 ChangeMenu(hmnSystem,Ø,NULL,Ø,MF_APPEND|MF_SEPARATOR);
 LoadString(hiMain,STTURNOFF,zBuf,sizeof(zBuf));
 ChangeMenu(hmnSystem,Ø,zBuf,CMD_TURNOFF,MF_APPEND|MF_STRING);
 LoadString(hiMain,STMODE,zBuf,sizeof(zBuf));
 ChangeMenu(hmnSystem,Ø,zBuf,CMD_MODE,MF_APPEND|MF_STRING);

 /* Show application always as icon */
 ShowWindow(hwMain,SW_SHOWMINNOACTIVE);

 /* Display mode dialog box if non-iconic show mode */
 if (vCmdShow!=SW_SHOWMINNOACTIVE)
  {v=DialogBox(hiMain, MAKEINTRESOURCE(DB_MODE), hwMain,
                                                  rfdMode);
   if (v==-1) ErrorNoMem();
  } /* if */
 SetMode(FALSE,bSaveActive); /* define active-screen mode */
 /* return without error */
 return TRUE;

 MemError:
 ErrorNoMem(); return FALSE; /* return with error */
} /* InitInstance() */


/****************************************************************

  --------------------- Main function -----------------------

****************************************************************/

WORD pascal WinMain(hiNew,hiPrev,zCmdLine,vCmdShow)
  HANDLE hiNew,hiPrev;
  LPSTR  zCmdLine;
  int    vCmdShow;

 {MSG wm;

  /* Initialize (window and) instance, return if error */
  if (!InitInstance(hiNew,hiPrev,zCmdLine,vCmdShow))
   return FALSE;
  /* --- application execution loop --- */
  while (GetMessage(&wm,NULL,Ø,Ø))
   {TranslateMessage(&wm); DispatchMessage(&wm);
   } /* while */
  return wm.wParam;
 } /* WinMain() */


/* ================================
   End of WINDOWS Application SCRSAVE
   ================================
*/
```

Bild 3: *Das C-Programm der Applikation* `SCRSAVE.C`

Marcellus Buchheit *ist Geschäftsführer der* Buchheit software research, *einem Unternehmen, das sich auf die Entwicklung von Anwendungen unter Microsoft Windows (und in Zukunft auch dem OS/2-Presentation Manager) spezialisiert hat.* Buchheit software research *berät auch Firmen, wie unter Windows programmiert wird und welche neue Perspektiven ein solches System dem Programmierer bietet. Außerdem entwickelt die Firma im Auftrag Software für Windows.*

```
/*************************************************************

 ----- Resource file for MS-Windows Application "SCRSAVE" -----

 ==============================================================

Copyright 1988 by
        Marcellus Buchheit, Buchheit software research
        Zaehringerstrasse 47, D-7500 Karlsruhe 1
        Phone (0) 721/37 67 76    (West-Germany)

All rights reserved --- Release 1.00 of 88-Aug-01

*************************************************************/

#include <WINDOWS.H>
#include "DEFS.H"


SCRSAVE ICON SCRSAVE.ICO
STRINGTABLE BEGIN
 STTITLE,"Bildschirm aus"
 STMODE,"&Einstellungen..."
 STTURNOFF,"Bildschirm &aus"
 STERROR,
 "Zu wenig Speicherplatz. Bitte beenden Sie eine Applikation."
 STTIMEOVL,"Zu viele Zeitgeber gestartet."
 ST2NDINST,"Applikation bereits gestartet."
END /* STRINGTABLE */

DB_MODE DIALOG LOADONCALL MOVEABLE DISCARDABLE 0,-76,112,76
CAPTION "Bildschirm aus"
STYLE WS_BORDER|WS_CAPTION|WS_DLGFRAME|WS_POPUP
BEGIN
 CONTROL "r 1988, Marcellus Buchheit",IDNONE,"static",
         SS_LEFT|WS_CHILD,4,4,104,8
 CONTROL "&Aktiv",IDACTIVE,"button",
         BS_AUTOCHECKBOX|WS_TABSTOP|WS_GROUP|WS_CHILD,
         4,20,32,12
 CONTROL "&Verz+gerung:",IDNONE,"static",
         SS_LEFT|WS_CHILD,4,40,48,8
 CONTROL "",IDDELAY,"edit",
         ES_LEFT|WS_BORDER|WS_TABSTOP|WS_GROUP|WS_CHILD,
         56,38,16,12
 CONTROL "min",IDNONE,"static",
         SS_LEFT|WS_CHILD,74,40,12,8
 CONTROL "&OK",IDOK,"button",
         BS_DEFPUSHBUTTON|WS_TABSTOP|WS_GROUP|WS_CHILD,
         4,58,20,14
 CONTROL "&Beenden",IDEXIT,"button",
         BS_PUSHBUTTON|WS_TABSTOP|WS_CHILD,72,58,36,14
END /* DB_MODE */
```

Bild 4: *Die Resource-Datei* SCRSAVE.RC

```
NAME    SCRSAVE
DESCRIPTION 'Screen Save Application - 1.00'
STUB    'WINSTUB.EXE'
CODE    FIXED
DATA    FIXED

HEAPSIZE  512
STACKSIZE 4096

EXPORTS
 fwMain  @1
 fdMode  @2
 fhEvent @3
```

Bild 5: *Die Linker-Definitionsdatei* SCRSAVE.DEF

```
SCRSAVE.RES: SCRSAVE.RC DEFS.H SCRSAVE.ICO
 RC -r SCRSAVE.RC

SCRSAVE.OBJ: SCRSAVE.C DEFS.H
 CL >SCRSAVE.ERR -c -Answ -Gsw -Os -Zdp SCRSAVE.C

SCRSAVE.EXE: SCRSAVE.OBJ SCRSAVE.RES SCRSAVE.DEF
 LINK4 /A:16/M/LI SCRSAVE,,,SLIBW,SCRSAVE
 MAPSYM SCRSAVE
 RC SCRSAVE.RES
```

Bild 6: *Die Make-Datei* SCRSAVE

```
/* dialog box resources */
#define DB_MODE            100

/* menu commands */
#define CMD_TURNOFF  200
#define CMD_MODE     201
#define CMD_ABOUT    202

/* strings */
#define STTITLE            1000
#define STERROR            1001
#define ST2NDINST    1002
#define STMODE             1003
#define STTURNOFF    1004
#define STTIMEOVL    1005

/* dialog box entries */
#define IDNONE             0
#define IDACTIVE     2000
#define IDDELAY            2001
#define IDEXIT             2002
```

Bild 7: *Die Header-Datei* DEFS.H

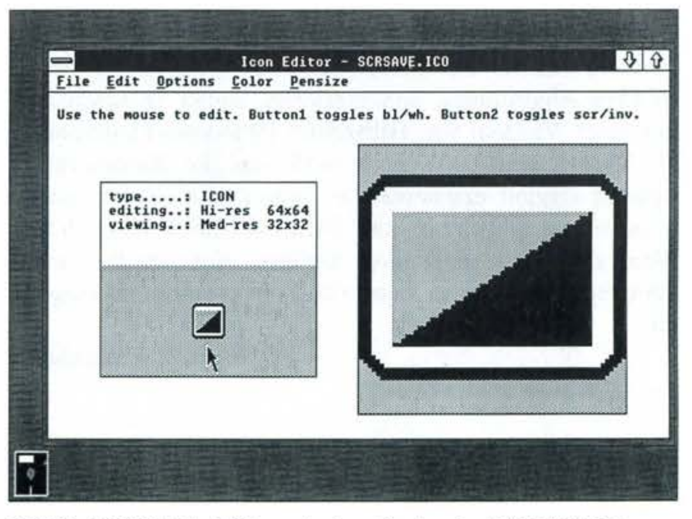


Bild 8: SCRSAVE.ICO *nach dem Laden in ICONEDIT*

Eingriffs-
funktion
(Hook function)
der
Applikation

Nachrichtenverarbeitung
Applikation

Applikation

Nachrichten-
erzeugung
"Eingabe"

Weg bei Eingriff
(Hook)

normaler Weg

Nachrichtensammlung
MS-Windows

Windows

Tastatur-
dekoder

Maus
dekoder

Treiber

Tastatur

Maus

Hardware

Bild 9: *Windows-Kern-Eingriff (Windows-Hook) für die Eingabe-Nachrichten*

Literatur

[1] *David Durant, Geta Carlson, Paul Yao: Programmer's Guide to WINDOWS, 2nd Edition, 1988, Sybex San Francisco, Paris, Düsseldorf, London.*

[2] *Charles Petzold: Programming WINDOWS, First Edition, 1988, Microsoft Press*

[3] *Charles Petzold: Anzeige des freien Speichers unter Windows, Microsoft System Journal Juli/August 1988, Seite 12 bis 18.*

Dialoge ganz nebenbei:

Dynamische Dialogboxen unter Windows

Dialogboxen oder Eingabemasken dynamisch, so ganz »nebenbei«, zu erzeugen, war unter Windows 1.x nicht möglich. Der einzige Weg, eine Eingabemaske oder Dialogbox zu definieren, bestand in der Ressourcendatei. Microsoft Windows 2.x enthält nun jedoch zwei neue Funktionen, mit denen Programmierer dynamisch solche Dialogboxen erzeugen können, wie sie in den Beispielmakros von Microsoft Excel zu sehen sind.

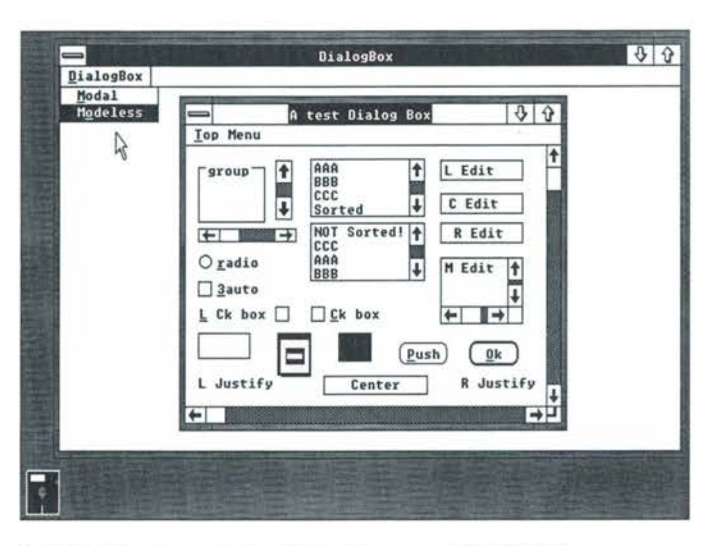


Bild 1: *Die dynamische Dialogbox von DLGBOX.*

DialogBoxIndirect und CreateDialogIndirect

Die beiden neuen Funktionen von Windows 2.0, die hier gemeint sind, heißen `DialogBoxIndirect` und `CreateDialogIndirect`. Eine genaue Untersuchung zeigt, daß der einzige Unterschied zwischen den Standardfunktionen `DialogBox/CreateDialog` und den Funktionen `DialogBoxIndirect/CreateDialogIndirect` darin besteht, daß letztere anstelle eines Long-Pointers eine Handle als zweiten Parameter verwenden. Wenn man diese Routinen in einer Anwendung einsetzt, hat das Programm oder der Benutzer die Möglichkeit, Dialogboxen nach Wunsch anzulegen oder zu verändern.

Eine Dialogbox mit ungefähr 20 verschiedenen Feldern oder Steuerungen ist in *Bild 1* zu sehen. Das ungewöhnliche an dieser Dialogbox ist, daß sie dynamisch durch den Aufruf von `DialogBoxIndirect` oder `CreateDialogIndirect` angelegt wurde und keine Zeilen in der Ressourcendatei oder einen Aufruf der Funktion `CreateWindow` erforderte. Eine Tabelle für die Funktionen `DialogBoxIndirect` oder `CreateDialogIndirect` ist die einzige Voraussetzung.

Die Dialoge von Microsoft Excel

Eine der vielen leistungsfähigen Möglichkeiten von Microsoft Excel ist, daß der Benutzer eigene Dialogboxen definieren kann. Das erfolgt so, daß man eine Tabelle der gewünschten Steuerungen in einem Makroblatt definiert und diese Tabelle aufruft. Der gleiche Tabellentyp wird in Windows-Anwendungen benötigt; diese Tabelle kann jedoch jede beliebige Form und jedes beliebige Aussehen haben, die Sie für Ihre Anwendung wünschen. Sie kann eine Textdatei sein, die in das Programm eingelesen oder direkt eingebaut wird oder auch eine Parameterliste, die dadurch angelegt wird, daß der Benutzer Werte in eine Eingabemaske eingibt. Ganz gleich, wie die Tabelle aussieht, Ihr Programm kann dann Dialogboxen dynamisch anlegen und verändern.

Das Programm DlgBox

Das Programm DlgBox besteht aus zwei Modulen, `DLGBOX.C` und `DLGTEMP.C`. DlgBox ist der Hauptteil des Programms, von dem aus ein Aufruf des Moduls DlgTemp für jedes Dialogfeld erfolgt, daß sich in einer Dialogbox befinden soll. DlgTemp baut diese Felder in eine Tabelle ein, die später wieder an das Programm DlgBox zurückgegeben wird. Das Modul DlgTemp wurde als separate Einheit konzipiert, damit Sie es in Ihre Anwendung integrieren können. Wir wollen uns nun die verschiedenen Teile des Programms ansehen. Die Make-Datei (*Listing 1*) enthält alle Befehle, die notwendig sind, um diese Anwendung zu erstellen. Denken Sie daran, daß zwei Funktionen nur in Windows 2.0 und neueren Versionen verfügbar sind, Sie müssen also das Microsoft Windows Software Development Kit Version 2.x und den Microsoft C 5.0 Compiler oder eine neuere Version verwenden.

Die Ressourcendatei, die in *Listing 2* zu sehen ist, enthält das Sinnbild und zwei Menüs. Mit dem ersten Menü können Sie auswählen, ob die erzeugte Dialogbox modal oder moduslos ist. Der Unterschied wird später noch erklärt. Das zweite Menü wird in der dynamischen Dialogbox als Beispiel dafür verwendet, wie man es zu einem Teil der Dialogbox machen kann.

Neue Datenstrukturen

Bei der Verwendung von `DialogBoxIndirect` und `CreateDialogIndirect` werden für den Aufbau einer Dialogbox nur zwei Strukturen benötigt. Die erste ist die Struktur `DialogHeader` (*Listing 4*). Sie wird als Header der zu erstellenden Tabelle verwendet. Die zweite Datenstruktur ist `DialogItem` (*Listing 4*). Die Struktur `DialogItem` wird für jede Steuerung verwendet, die in der Dialogbox plaziert wird.

```
dlgtemp.obj: dlgtemp.c  dlgbox.h
     cl -d  -c  -W2 -AS  -Gsw  -Os -Zpe dlgtemp.c

dlgbox.obj: dlgbox.c  dlgbox.h
     cl -d  -c  -W2 -AS  -Gsw  -Os -Zpe dlgbox.c

dlgbox.res: dlgbox.rc dlgbox.h
     rc -r dlgbox.rc

dlgbox.exe:  dlgbox.obj dlgbox.def dlgbox.res dlgtemp.obj
     link4 dlgbox dlgtemp,dlgbox/al:16,dlgbox,
                                        slibw slibcew /noe,dlgbox.def
     rc dlgbox.res
```

Listing 1: *Die Make-Datei für DlgBox.*

```
#include <style.h>
#include "dlgbox.h"

dlgicon ICON dlgbox.ico

dlgbox MENU
BEGIN
  POPUP "&DialogBox"
  BEGIN
    MENUITEM "&Modal",     IDM_MODAL
    MENUITEM "M&odeless", IDM_MODELESS
  END
END

dlgtemp MENU
BEGIN
  POPUP "&Top Menu"
  BEGIN
    MENUITEM "&One", IDM_ONE
  END
END
```

Listing 2: *Die Ressourcen-Datei für DlgBox.*

```
NAME     Dlgbox

STUB     'WINSTUB.EXE'

CODE     MOVEABLE
DATA     MOVEABLE MULTIPLE

HEAPSIZE    4096
STACKSIZE   4096

EXPORTS
    DlgboxWndProc           @1
    DialogBoxWindowProc     @2
```

Listing 3: *Die Moduldefinitionsdatei für DlgBox.*

Nur drei Funktionen brauchen im Modul DlgTemp aufgerufen werden, um die ganze Arbeit für die Anlage der Dialogstruktur im globalen Speicherbereich durchzuführen. Dies sind die Funktionen `CreateDialogHeader`, `CreateDialogItem` und `EndDialogHeader`.

CreateDialogHeader

Die Funktion `CreateDialogHeader` übergibt Parameter zum Auffüllen der verschiedenen Teile der Struktur `DialogHeader`, wie das in *Listing 5* zu sehen ist. Wenn die Struktur fertig aufgebaut ist, wird ein wenig globaler Speicher allokiert, um die `DialogHeader`-Informationen für die spätere Verwendung zu speichern.

Wenn Sie sich die Struktur `DialogHeader` ansehen, sehen Sie, daß sie aus den meisten Parametern besteht, die für die Struktur `CREATESTRUCT` von `CreateWindows` benötigt werden. Und genau dafür wird sie auch später, wenn die Felder ausgefüllt sind, eingesetzt. Das Feld `dtItemCount` ist das einzige neue. Es definert die Anzahl der Felder oder Steuerungen in der Dialogbox. Dieses Feld kann jedoch erst ausgefüllt werden, wenn die Anzahl der Felder bekannt ist. Das erste Feld, `dtStyle`, enthält die Fensterstilattribute für die Dialogbox bzw. `CreateWindow` (s. Tabelle 3.4 im Microsoft Windows Software Development Kit Programmer's Reference Version 2.x).

Die Felder `dtX` und `dtY` werden dazu verwendet, um die obere linke Ecke der Dialogbox im Zeichenbereich des übergeordneten Fensters zu positionieren. Die Felder `dtCX` und `dtCY` werden dann verwendet, um die Breite und Höhe der Dialogbox zu definieren. Um festzustellen, wie breit die Dialogbox sein sollte, multiplizieren Sie die Anzahl Zeichen mit 4 und addieren etwa 10 hinzu, um einen Rand auf beiden Seiten zu erhalten. Für die Höhe multiplizieren Sie die Anzahl Zeilen mit 8 und addieren auch wieder 10 für den Rand darauf. Der Einfachheit halber können auch die Breite und Höhe vom Dialog-Editor `DIALOG.EXE` direkt und ohne Umwandlung verwendet werden.

Der folgende Teil der Struktur besteht aus drei Feldern mit variabler Länge. Sie werden variabel genannt, da die Größe des Feldes zwischen 1 und vielen Byte schwanken kann. Da wir nicht im voraus wissen, wie lang diese Felder sein werden, können wir keinen Speicherbereich dafür zuweisen. Wir könnten Sie so lang wie möglich definieren, doch dadurch wird unnötig Platz verschwendet. Bei der Verwendung dieser Felder wird eine einfache Regel befolgt: Wenn eines nicht gebraucht wird, ist das erste Byte 0.

Im ersten Byte nach der Struktur `DialogHeader` folgt der Ressourcen-Name, `dtResourceName`, der den Namen der Menü-Ressource in der Ressourcendatei der Anwendung definiert. Wenn das erste Byte 0 ist, wird bei dieser Dialogbox kein Menü verwendet. Wenn Sie die Nummer der Menü-Ressource verwenden wollen, muß das erste Byte FFh sein und darauf in drei Byte die Zahl (in ASCII) folgen. Wenn das erste Byte nicht 0 oder FFh ist, wird ein mit 0 abgeschlossener String erwartet.

Das Feld `dtClassName` definiert die Fensterklasse der Dialogbox. Wenn das erste Byte 0 ist, wird die Standardklasse `DialogClass` verwendet. Wenn Sie Ihre eigene Fensterklasse definieren wollen, übergeben Sie einen mit 0

abgeschlossenen String in `dtClassName`. Dies ist der gleiche Wert, der im Feld `lpWindowName` der Funktion `CreateWindow` verwendet wird.

Nun da wir mit dem Füllen der Struktur `DialogHeader` fertig sind, haben wir alle Felder für den Aufruf der Funktion `CreateWindow` beisammen. In unserem Beispiel haben wir das vereinfacht, indem wir die Parameter der Funktion `CreateDialogHeader` übergeben haben. Wir können nun die einzelnen Felder angeben, die in der Dialogbox angezeigt werden sollen.

CreateDialogItem

Die nächste Funktion in unserem Programm ist `CreateDialogItem`. Mit ihr werden alle Informationen übergeben, die notwendig sind, um die verschiedenen Teile der Struktur `DialogItem` aufzufüllen. Nachdem die Struktur gefüllt ist, wird der globale Speicherblock mit der Funktion `GlobalRealloc` auf die benötigte Größe erweitert. Dann werden die Informationen in den erweiteren Speicherblock kopiert.

Ein Vorteil der Funktion `CreateDialogItem` ist, daß die meisten Fenstersteuerungen und ihre Stile bereits vordefiniert sind, so daß man sie mit einer Zahl ansprechen kann. Dies erspart es einem, all die verschiedenen Attribute für eine vordefinierte Steuerung übergeben zu müssen, wie in Microsoft Excel.

Die Struktur `DialogItem` besteht aus den gleichen Feldern, wie die Struktur `DialogHeader`, denn diese Werte werden benötigt, um die Funktion `CreateWindow` aufzurufen. Die Felder `dtilX`, `dtilY`, `dtilCX`, `dtilCY` definieren die obere linke Ecke und die Breite/Höhe der Steuerung. Das Feld `dtilID` ist die Steuerungs-IDs (z.B. IDOK). Dieser Wert ist der gleiche, wie er für das Feld `hMenu` der Funktion `CreateWindow` verwendet wird.

Das Verhalten der Steuerung wird mit dem Feld `dtilStyle` definiert. Es kann jeder der vordefinierten Windows-Stile sein und entspricht dem Wert, der für das Feld `hStyle` der Funktion `CreateWindow` verwendet wird. Zusätzlich können Sie alle gewünschten Fenster- und Steuerungsstilattribute damit logisch oder (`||`) verknüpfen. (Eine vollständige Liste finden Sie in den Tabellen 3.4 und 3.5 in Microsoft Windows Software Development Kit Programmer's Reference Version 2.x.)

Dann folgt `dtilControlClass` für die Definition der spezifischen Steuerungsklasse des Feldes. In *Listing 4* ist eine Liste der vordefinierten Klassen zu sehen. (Eine umfassendere Beschreibung jeder Steuerungsklasse bietet die Tabelle 3.3 in Microsoft Windows SDK Programmer's Reference Version 2.x.)

Das Feld `dtilText` wird für den Text verwendet, der in der Steuerung angezeigt werden soll. Wenn das erste Byte 0 ist, wird kein Text angezeigt; wenn das erste Byte nicht 0 ist, wird ein mit 0 abgeschlossener String erwartet.

```
#define IDM_MODAL      100
#define IDM_MODELESS   110
#define IDM_ONE        120

#define IDDEDIT        3025
#define IDCHECK        3026

/* Control class codes */
#define BUTTONCLASS    0x80 /* 128 */
#define EDITCLASS      0x81 /* 129 */
#define STATICCLASS    0x82 /* 130 */
#define LISTBOXCLASS   0x83 /* 131 */
#define SCROLLBARCLASS 0x84 /* 132 */

typedef struct DialogHeader{
   long  dtStyle;
   BYTE  dtItemCount;
   int   dtX;
   int   dtY;
   int   dtCX;
   int   dtCY;
} DLGHDR;

/* The following 3 header items are variable strings
   so not part of DLGHDR */

/*   char dtResourceName[]; */
/*   char dtClassName[];    */
/*   char dtCaptionText[];  */

typedef struct DialogItem {
   int   dtilX;
   int   dtilY;
   int   dtilCX;
   int   dtilCY;
   int   dtilID;
   long  dtilStyle;
   BYTE  dtilControlClass;
} DLGITEM;

/* The following control items are variable
   so not part of DLGITEM */

/*   char dtilText[]; */
/*   BYTE dtilInfo; */

#define  DI0   0L
#define  DI1   1L
#define  DI2   2L
#define  DI3   3L
#define  DI4   4L
#define  DI5   5L
#define  DI6   6L
#define  DI7   7L
#define  DI8   8L
#define  DI9   9L
#define  DI10  10L
#define  DI11  11L
#define  DI12  12L
#define  DI13  13L
#define  DI14  14L
#define  DI15  15L
#define  DI16  16L
#define  DI17  17L
#define  DI18  18L
#define  DI19  19L
#define  DI19  19L
#define  DI20  20L
#define  DI21  21L
```

Listing 4: *Die Header-Datei* `DLGBOX.H`.

Das Feld `dtilInfo` entspricht dem letzten Feld der Struktur `WNDCLASS`, `cbWndExtra`. Es ermöglicht es uns, zusätzliche Bytes an die Struktur `WNDCLASS` zu hängen. Wir können diese zusätzlichen Informationen dann vom Fenster mit den Funktionen `GetWindowWord`, `GetWindowLong`,

SetWindowWord und SetWindowLong abfragen oder speichern. Sobald wir alle Informationen zusammen haben, können wir sie als Parameter der Funktion CreateDialogItem übergeben. Wenn alles richtig vonstatten geht, ist der globale Speicher erweitert, die neue zusätzliche Information an das Ende kopiert und der Feldzähler (iItems) um eins hochgezählt worden. Dadurch brauchen wir nach Übergabe des letzten Steuerungsfeldes in der Dialogbox die Felder in der Dialogbox nicht noch einmal durchzuzählen.

EndDialogHeader

Die letzte Funktion, die in unserer Beispielanwendung verwendet wird, ist EndDialogHeader. Hier müssen wir hergehen und das Feld dtItemCount in DialogHeader, das sich nun im globalen Speicher befindet, direkt ändern. Wir erledigen das, indem wir erst den Speicher locken und dann den Zeiger über die korrekte Anzahl Byte bringen. Dann ändern wir den Speicherwert auf die richtige Feldanzahl, die wir in iItems in der Funktion CreateDialogItem mitgezählt haben. Dann wird der Speicher wieder »entlockt«. Wenn die Funktion EndDialogHeader den Speicher ändern kann, übergibt sie eine Handle auf den globalen Speicher, der unsere neu angelegte Dialogtabelle enthält.

Jetzt, wo die Dialogstruktur aufgebaut ist, können wir die Dialogtabelle an eine unserer Dialogbox-Funktionen übergeben. Wenn wir eine modale Dialogbox wollen, brauchen wir nur die Funktion DialogBoxIndirect mit der Handle auf den globalen Speicher aufrufen. Wenn wir jedoch eine moduslose Dialogbox wollen, ist ein weiterer Schritt notwendig - wir müssen den globalen Speicher locken, damit wir einen Long-Pointer als zweiten Parameter der Funktion CreatDialogIndirect übergeben können. Außer dem Locken des Speichers besteht der Hauptunterschied zwischen DialogBoxIndirect und CreateDialogIndirect darin, daß DialogBoxIndirect auch die Funktion DisableWindow aufruft, um die Fensterhandle, die als dritter Parameter übergeben wurde, abzuschalten. Wenn diese Handle nicht die Handle des übergeordneten Fensters ist, findet die Funktion DialogBoxIndirect sie heraus und schaltet sie ab.

Abschluß

Dynamische Dialogboxen unterscheiden sich nicht stark von ihren statischen Gegenstücken. Die Möglichkeit, sie anzulegen wenn man sie benötigt, ist jedoch ein sehr nützliches Feature, das zusätzliche Fexibilität bringt. Nachdem Sie nun gelernt haben, wie Sie die Dialogtabelle aufbauen, die die beiden neuen Funktionen benötigen, und auch verstehen, was die Funktionen in DLGTEMP.C tun, haben Sie alle notwendigen Tools, um damit zu beginnen, Anwendungen zu erstellen, die Dialogboxen enthalten, die »nebenbei« entstanden sind.

Don Hasson

```
/* DLGBOX.C -- version 1.00 Demonstrates how to use the
            DialogBoxIndirect and CreateDialogIndirect
            functions.     */

#include <windows.h>
#include "dlgbox.h"

/* List of functions used in this module. */
BOOL FAR PASCAL InitDiabox(HANDLE, HANDLE, int);
LONG FAR PASCAL DlgboxWndProc(HANDLE, unsigned, WORD, LONG);
BOOL FAR PASCAL DialogBoxWindowProc(HANDLE,unsigned,WORD,LONG);
BOOL      BuildDialog(HWND);

/* The following functions are found in the DLGTEMP module.
   They are used to build the dialog structure that will be
   passed to the DialogBoxIndirect or CreateDialogIndirect
   function.
 */

/* -------- Functions defined in another module ------------ */
extern BOOL      FAR PASCAL CreateDialogHeader(LONG,BYTE,int,
                            int,int,int,LPSTR,LPSTR,LPSTR);
extern BOOL      FAR PASCAL CreateDialogItem(int,LONG,BYTE,int,
                            int,int,int,LPSTR,BYTE );
extern HANDLE    FAR PASCAL EndDialogHeader( );

/* -------- Static variables ----------- */
HANDLE   hInst;             /* handle to Instance          */
FARPROC  lpDlgTest;         /* Dialog processing routine   */
HWND     hDlgTest = NULL;   /* handle for modeless dlg box */
BOOL     nModeless;         /* flag set by menu selected   */

HANDLE   hDlgTemp;    /* Handle to created dialog template   */
LPSTR    lpDTemplate; /* Long pointer to dialog template     */

/* ------------------ WinMain ------------------------------ */
int PASCAL WinMain(hInstance,hPrevInstance,lpszCmdLine,cmdShow)
HANDLE  hInstance , hPrevInstance;
LPSTR   lpszCmdLine;
int     cmdShow;
{
   MSG msg;
   BOOL bResult;   /* Modeless dialog message processed flag */

   if (hPrevInstance) /* do not allow more than one instance */
      return FALSE;

   InitDiabox ( hInstance, hPrevInstance, cmdShow );

   while ( GetMessage((LPMSG)&msg, NULL, 0, 0 )) {
      bResult = FALSE;

      /* this structure allows checking for multiple
         modeless dialog boxes */

      if ( hDlgTest != NULL )
         bResult = IsDialogMessage( hDlgTest, (LPMSG)&msg );

      if ( !bResult ) {
         TranslateMessage((LPMSG)&msg);
         DispatchMessage((LPMSG)&msg);
      }
   }
   return msg.wParam ;
}

/* ------------- Initialization section ------------------- */

BOOL FAR PASCAL InitDiabox( hInstance, hPrevInstance, cmdShow )
HANDLE hInstance;
HANDLE hPrevInstance;
int    cmdShow;
{
   WNDCLASS wcDiaboxClass;
   HWND     hWnd;

   wcDiaboxClass.lpszClassName = (LPSTR) "Diabox";
   wcDiaboxClass.hInstance     = hInstance;
   wcDiaboxClass.lpfnWndProc   = DlgboxWndProc;
   wcDiaboxClass.hCursor       = LoadCursor( NULL, IDC_ARROW );
```

```
    wcDiaboxClass.hIcon          = (HICON)LoadIcon(hInstance,
                                          "dlgicon" );
    wcDiaboxClass.lpszMenuName  = (LPSTR)"dlgbox";/*menu added*/
    wcDiaboxClass.hbrBackground = GetStockObject(WHITE_BRUSH);
    wcDiaboxClass.style         = CS_HREDRAW | CS_VREDRAW;
    wcDiaboxClass.cbClsExtra    = 0;
    wcDiaboxClass.cbWndExtra    = 0;

    RegisterClass ( (LPWNDCLASS) &wcDiaboxClass );

    hWnd = CreateWindow( (LPSTR)"Diabox",
                         (LPSTR)"DialogBox",
                         WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT,
                         CW_USEDEFAULT,
                         CW_USEDEFAULT,
                         CW_USEDEFAULT,
                         (HWND)NULL,
                         (HMENU)NULL,
                         (HANDLE)hInstance,
                         (LPSTR)NULL
                       );

    hInst = hInstance;       /* instance saved for dialog box */

    ShowWindow( hWnd, cmdShow );
    UpdateWindow( hWnd );

    return( TRUE );
}

/* ------------ parent's window procedure ------------------ */

LONG FAR PASCAL DlgboxWndProc( hWnd, message, wParam, lParam )
HWND      hWnd;
unsigned message;
WORD      wParam;
LONG      lParam;
{
    PAINTSTRUCT ps;
    int    iResult;

    switch (message) {
       case WM_COMMAND:
          switch (wParam) {

             case IDM_MODAL:

                if ( hDlgTest == NULL ) {
                       /* allow only one dialog up at a time */
                  nModeless = FALSE;
                  if ( BuildDialog(hWnd) == TRUE) {
                    lpDlgTest = MakeProcInstance(
                                (FARPROC) DialogBoxWindowProc,
                                hInst );

                    if (lpDlgTest == NULL) {
                      MessageBox( GetFocus(),
                                  "MakeProcInstance failed!",
                                  "ERROR", MB_OK );
                      GlobalFree( hDlgTemp );
                      return FALSE;
                    }
                    iResult = DialogBoxIndirect(hInst, hDlgTemp,
                                               hWnd, lpDlgTest);
                  }
                }
                break;

             case IDM_MODELESS:

                if ( hDlgTest == NULL ) {
                       /* allow only one dialog up at a time */
                  nModeless = TRUE;
                  if ( BuildDialog(hWnd) == TRUE) {
                    lpDTemplate = GlobalLock( hDlgTemp );
                    if (lpDTemplate == NULL) {
                      MessageBox( GetFocus(),
                                  "Could not lock memory.",
                                  "Diabox", MB_OK );
                      GlobalFree( hDlgTemp );
                      return FALSE;
                    }
                    lpDlgTest = MakeProcInstance(
                                (FARPROC) DialogBoxWindowProc,
                                hInst );

                    if (lpDlgTest == NULL) {
                      MessageBox( GetFocus(),
                                  "MakeProcInstance failed!",
                                  "ERROR", MB_OK );
                      GlobalFree( hDlgTemp );
                      return FALSE;
                    }
                    hDlgTest = CreateDialogIndirect(hInst,
                               lpDTemplate, hWnd, lpDlgTest );
                  } /* exit if BuildDialog didn't work */
                }
                break;

             default:
                return DefWindowProc(hWnd,message,wParam,lParam);
                break;
          }

       case WM_PAINT:
          BeginPaint( hWnd, (LPPAINTSTRUCT)&ps );
          EndPaint  ( hWnd, (LPPAINTSTRUCT)&ps );
          break;

       case WM_DESTROY:
          FreeProcInstance ( (FARPROC)lpDlgTest );
          PostQuitMessage( 0 );
          break;

       default:
          return( DefWindowProc(hWnd,message,wParam,lParam));
    }

    return( 0L );
}


/* ------------- Build dialog routine ---------------------- */
/*
 *  This routine passes all the necessary information to the
 *  dialog functions in DLGTEMP.C that will build the dialog
 *  table.
 */

BOOL BuildDialog(hWnd)
HWND  hWnd;
{
    BOOL      bResult;
    HCURSOR hOldCursor;          /* Handle to old mouse cursor */

    /* Build the Dialog header */

    bResult = CreateDialogHeader(
      WS_BORDER   | WS_CAPTION | WS_DLGFRAME | WS_VSCROLL |
      WS_HSCROLL  | WS_SYSMENU | WS_GROUP    | WS_TABSTOP  |
      WS_SIZEBOX | WS_VISIBLE | WS_POPUP, /* window style */
      (BYTE)12, 24, 12, 180, 160, /*coordinates of Dialog box.*/
      "dlgtemp",                /* menu       */
      "",                       /* class name  */
      "A test Dialog Box" );    /* Caption bar */

    if (bResult == FALSE) {
      MessageBox( GetFocus(),
                  "Not enough memory.",
                  "Diabox", MB_OK );
      return FALSE;
    }

    hOldCursor = SetCursor( LoadCursor(NULL, IDC_WAIT) );

    /* Each call to CreateDialogItem is an item to be placed  */
    /* inside the dialog box                                  */
    /* This next comment line is the value that is passed     */
    /* id,  style, class, x, y, cx, cy,text, extrabytes       */
```

```
   CreateDialogItem(  IDCHECK,DI0,   0,
              60,  90, 45, 12,  "&Ck box",0x00 );/* ck box   */
   CreateDialogItem( 3001,    DI1,   0,
              44, 107,  0,  0,  "dlgicon",0x00 );/* icon     */
   CreateDialogItem( 3002,    DI2,   0,
              73, 107, 16, 17,         "",0x00 );/* black box*/
   CreateDialogItem( 3003,    DI3,   0,
               6, 107, 25, 15,         "",0x00 );/* rect     */
   CreateDialogItem( 3004,    DI4,   0,
               6, 132, 40, 10,"L Justify",0x00 );/* lft text */
   CreateDialogItem( 3005,    DI5,   0,
             122,  64, 40, 38,   "M Edit",0x00 );/* muli edt */
   CreateDialogItem( 3006,    DI6,   0,
              60,   8, 54, 32,       "lb",0x00 );/* list box */
   CreateDialogItem( 3007,    DI7,   0,
              43,   8,  8, 34,      "v s",0x00 );/*vert scrl */
   CreateDialogItem( 3008,    DI8,   0,
               6,  47, 47,  8,      "h s",0x00 );/*horz scrl */
   CreateDialogItem( 3009,    DI9,   0,
               6,   8, 32, 36,    "group",0x00 );/* group box*/
   CreateDialogItem( IDCANCEL,DI10,  0,
             102, 112, 24, 14,    "&Push",0x00 );/*push but  */
   CreateDialogItem( 3011,    DI11,  0,
               6,  61, 45, 12,   "&radio",0x00 );/*radio but */
   CreateDialogItem(     IDOK,DI12,  0,
             136, 112, 24, 14,      "&Ok",0x00 );/*def buton */
   CreateDialogItem( 3013,    DI13,  0,
               6,  90, 44, 12,"&L Ck box",0x00 );/*left ckbox*/
   CreateDialogItem( 3014,    DI14,  0,
               6,  76, 30, 12,   "&3auto",0x00 );/*3 autoton */
   CreateDialogItem( 3015,    DI15,  0,
             122,  27, 40, 12,   "C Edit",0x00 );/*edit cent */
   CreateDialogItem( 3016,    DI16,  0,
             122,  44, 40, 12,   "R Edit",0x00 );/*rt edit   */
   CreateDialogItem(  IDDEDIT,DI17,  0,
             122,   8, 40, 12,   "L Edit",0x00 );/* L edit   */
   CreateDialogItem( 3018,    DI18,  0,
              60,  45, 54, 32,  "no sort",0x00 );/* lb w/out */
   CreateDialogItem( 3019,    DI19,  0,
              66, 132, 50, 10,   "Center",0x00 );/* CEN text */
   CreateDialogItem( 3020,    DI20,  0,
             122, 132, 46, 10,"R Justify",0x00 );/* RIG text */

   SetCursor( hOldCursor );

   /* Get the handle to the new dialog table */

   /* end dialog template, return hDlgTemp. */
   hDlgTemp = EndDialogHeader();
   if (hDlgTemp == NULL) {
     GlobalFree(hDlgTemp);
     MessageBox( GetFocus(),
               "End dialog routine.",
               "Error!", MB_OK );
     return FALSE;
   }
   return TRUE;
}

/* -------------- dialog box window procedure ------------- */
/*
 *  This routine processes all the message for the dialog that
 *  was dynamically created.
 */

BOOL FAR PASCAL DialogBoxWindowProc(hDlg,message,wParam,lParam)
HWND     hDlg;
unsigned message;
WORD     wParam;
LONG     lParam;
{
   char     TempEdit[24];
   HWND     hListBox, hListBoxNS;

   switch(message) {

      case WM_INITDIALOG:

         hListBox = GetDlgItem( hDlg, 3006 );

         SendMessage(hListBox, LB_ADDSTRING, 0,
                     (LONG)(LPSTR)"Sorted");
         SendMessage(hListBox, LB_ADDSTRING, 0,
                     (LONG)(LPSTR)"CCC");
         SendMessage(hListBox, LB_ADDSTRING, 0,
                     (LONG)(LPSTR)"AAA");
         SendMessage(hListBox, LB_ADDSTRING, 0,
                     (LONG)(LPSTR)"BBB");

         hListBoxNS = GetDlgItem( hDlg, 3018 );
         SendMessage(hListBoxNS, LB_ADDSTRING, 0,
                     (LONG)(LPSTR)"NOT Sorted!");
         SendMessage(hListBoxNS, LB_ADDSTRING, 0,
                     (LONG)(LPSTR)"CCC");
         SendMessage(hListBoxNS, LB_ADDSTRING, 0,
                     (LONG)(LPSTR)"AAA");
         SendMessage(hListBoxNS, LB_ADDSTRING, 0,
                     (LONG)(LPSTR)"BBB");

         SetFocus( GetDlgItem( hDlg, IDOK ) );
         return(TRUE);

         break;

      case WM_COMMAND:

         switch( wParam ) {

            case IDCHECK: /*  Check Box      */
               CheckDlgButton(hDlg, IDCHECK,
                              !IsDlgButtonChecked(hDlg,IDCHECK));
               break;

            case IDM_ONE: /*  One menu item  */

               MessageBox(GetFocus(), (LPSTR)"One selected.",
                              (LPSTR)"Menu item", MB_OK );
               break;

            case IDOK:     /*  Ok button      */

               GetDlgItemText( hDlg, IDDEDIT, TempEdit, 24 );
               MessageBox(GetFocus(), (LPSTR)TempEdit,
                              (LPSTR)"Edit:", MB_OK );
               /* fall through to IDCANCEL */

            case IDCANCEL: /* Cancel push button */

               hDlgTest = NULL;
               if ( nModeless == FALSE )
                  EndDialog ( hDlg, TRUE );
               else
                  DestroyWindow( hDlg );
               return( TRUE );

            default:
                return( TRUE );
         }  /* end wParam switch */
         break;   /* end wm_command */

      case WM_DESTROY:

         hDlgTest = NULL;

         while (GlobalFlags(hDlgTemp) & GMEM_LOCKCOUNT)
             GlobalUnlock( hDlgTemp );
         if (hDlgTemp)
           GlobalFree( hDlgTemp );

         break;

      default:
         return( FALSE );

   } /* end message switch */
   return( FALSE );
}
```

Listing 5: *Die C-Quelldatei* `DLGBOX.C`.

```
/* DLGTEMP.C -- version 1.00
          This module supplies the necessary functions
          used to create a dialog box template in global
          memory and to pass the handle on to either the
          DialogBoxIndirect or CreateDialogIndirect functions.
 */

#include <windows.h>
#include "dlgbox.h"

/*  forward references */
WORD    lstrlen( LPSTR );
void    SetStyleClass(int);

HANDLE   hDlgTemplate; /* Handle to dialog template memory */
WORD     wOffset;      /* Current memory offset
                          (updated by CDH & CDI) */
BYTE     iItems;       /* number of items in dialog */
DLGITEM  DlgItem;      /* Dialog item structure */

/* ------------- Create Dialog Header ----------------------*/
/*
 *  This routine allocates a piece of global memory
 *  and then fills in the dialog header structure and saves the
 *  information in global memory.
 */

BOOL FAR PASCAL CreateDialogHeader( Style, ItemCount, X, Y,
                                    cX, cY, Resource, Class, Caption)
LONG  Style;     /* Dialog box Style                          */
BYTE  ItemCount; /* Control count for dialog box              */
int   X;         /* Dialog box top left column                */
int   Y;         /* Dialog box top row                        */
int   cX;        /* Dialog box width                          */
int   cY;        /* Dialog box height                         */
LPSTR Resource;  /* Dialog box resource string                */
LPSTR Class;     /* Dialog box class string                   */
LPSTR Caption;   /* Dialog box caption                        */
{

   WORD   ResourceLength, /* Length of resource string        */
          ClassLength,    /* Length of class string           */
          CaptionLength;  /* Length of caption String         */
   DWORD  dwMemLength;    /* Dialog header memory allocation
                                                       length */
   LPSTR  lpHdrData;      /* Long pointer to locked dialog
                                              template memory */
   DLGHDR DlgHdr;         /* Dialog header structure          */
   LPSTR  lpDlgHdr;       /* Long pointer to dialog header
                                                    structure */
   int    i;              /* Loop index                       */

   /* Initialize memory offset */

   wOffset = 0;  /* set memory offset to ZERO */
   iItems  = 0;  /* set number of items in dialog to ZERO */

   /* Determine string lengths (including terminating null ) */

   ResourceLength = lstrlen( Resource ) + 1;
   ClassLength    = lstrlen( Class ) + 1;
   CaptionLength  = lstrlen( Caption ) + 1;

   /*Determine length of memory to allocate for dialog header*/
   dwMemLength = (DWORD)( sizeof( DLGHDR ) + ResourceLength +
                          ClassLength + CaptionLength );


   /* Allocate dialog template memory for dialog header
      and obtain handle */

   if ( !( hDlgTemplate = GlobalAlloc(GMEM_MOVEABLE |
                                      GMEM_ZEROINIT,
                                      dwMemLength ) ) ) {
      MessageBox( GetFocus(),
                  (LPSTR)"GlobalAlloc Error",
                  (LPSTR)"CDH", MB_OK );

      return FALSE; /* GlobalAlloc failed */
   }

   /* Lock allocated memory for modification */
   if ( lpHdrData = GlobalLock( hDlgTemplate ) ) {

      /* Set dialog header structure data to passed in parms */
      DlgHdr.dtStyle     = Style;
      DlgHdr.dtItemCount = 1;      /* set to zero */
      DlgHdr.dtX         = X;
      DlgHdr.dtY         = Y;
      DlgHdr.dtCX        = cX;
      DlgHdr.dtCY        = cY;

      /* Get pointer to dialog header structure */
      lpDlgHdr = (LPSTR)&DlgHdr;

      /* Copy dialog header structure to allocated memory */
      for ( i = 0; i < sizeof( DLGHDR ); i++ )
         *lpHdrData++ = *lpDlgHdr++;

      /* Copy resource string to allocated memory */
      while ( ( *lpHdrData++ = *Resource++ ) );

      /* Copy class string to allocated memory */
      while ( ( *lpHdrData++ = *Class++ ) );

      /* Copy caption string to allocated memory */
      while ( ( *lpHdrData++ = *Caption++ ) );

      /* Adjust memory offset past memory allocated for dialog
         header */
      wOffset += (WORD)( sizeof( DLGHDR ) + ResourceLength +
                         ClassLength + CaptionLength );

      /* Unlock allocated memory */
      GlobalUnlock( hDlgTemplate );

      return TRUE;  /* everything worked so far, return TRUE */
   }
   else {
      MessageBox( GetFocus(),
                  (LPSTR)"GlobalLock Error",
                  (LPSTR)"CreateDialogHeader", MB_OK );
      GlobalFree( hDlgTemplate ); /* free allocated memory */
      return FALSE;               /* return null handle
                                     indicating failure */
   }
}

/* ------------- Create Dialog Item ------------------------*/
/*
 *  This routine fills in the dialog item structure and
 *  saves the information in global memory after resizing it.
 */

BOOL FAR PASCAL CreateDialogItem( iCtrlID, lStyle, Class, X, Y,
                                  cX, cY, Text, ExtraBytes )
int    iCtrlID;       /* Control ID              */
LONG   lStyle;        /* Control style           */
BYTE   Class;         /* Control class           */
int    X;             /* Control top left column */
int    Y;             /* Control top row         */
int    cX;            /* Control width           */
int    cY;            /* Control height          */
LPSTR  Text;          /* Control text            */
BYTE   ExtraBytes;    /* Control extra bytes count */
{
  LPSTR    lpCtrlData;  /* Long pointer to locked dialog
                                             template memory */
  LPSTR    lpDlgItem;   /* Long pointer to dialog control
                                                   structure */
  int      i;           /* Loop index                        */
  WORD     TextLength;  /* Length of control text string     */
  DWORD    dwMemLength;
  HANDLE   hCurDlgTemp;


  DlgItem.dtilX  = X;
  DlgItem.dtilY  = Y;
  DlgItem.dtilCX = cX;
  DlgItem.dtilCY = cY;
  DlgItem.dtilID = iCtrlID;
```

```
  if (Class == (BYTE)0)
    SetStyleClass( (int)lStyle);
  else {               /* default, may change in next function */
      DlgItem.dtilControlClass = Class;
      DlgItem.dtilStyle = lStyle;
  }
  TextLength  = lstrlen( Text ) + 1;
  dwMemLength = (DWORD)(wOffset + sizeof(DLGITEM) +
                         TextLength + sizeof(BYTE) );

  hCurDlgTemp = GlobalReAlloc( hDlgTemplate, dwMemLength,
                               GMEM_MOVEABLE);

  if ( hCurDlgTemp == NULL) {
    MessageBox ( GetFocus(),
                 (LPSTR)"global lock",
                 (LPSTR)"Failed!", MB_OK);
    GlobalFree( hCurDlgTemp );
    return FALSE;
  }
  hDlgTemplate = hCurDlgTemp;

   /* Adjust pointer to reallocated memory bypassing
      existing data */

   if ( (lpCtrlData = GlobalLock(hDlgTemplate)) == NULL) {
     MessageBox ( GetFocus(),
                  (LPSTR)"global lock",
                  (LPSTR)"Failed!", MB_OK);
     GlobalFree( hDlgTemplate );
   }

   lpCtrlData += wOffset;

   /* Get pointer to dialog control structure */
      lpDlgItem = (LPSTR)&DlgItem;

   /* Copy dialog control structure to allocated memory */
      for ( i=0; i<sizeof( DLGITEM ); i++ )
         *lpCtrlData++ = *lpDlgItem++;

   /* Copy control test string to allocated memory */
      while ( ( *lpCtrlData++ = *Text++ ) );

   /* Copy extra byte count to allocated memory */
      *lpCtrlData = ExtraBytes;

   /* Adjust memory offset past memory reallocated for
      dialog control */
      wOffset += (WORD)( sizeof( DLGITEM ) +
                 TextLength + sizeof( BYTE ) );

   /* Unlock reallocated memory */
      GlobalUnlock( hDlgTemplate );

      iItems++;          /* bump up number of items in dialog */

      return( TRUE );    /* return successful */
}

/* ------------- End Dialog Header ------------------------- */
/*
 * This routine changes the number of items in the
 * dialog header and then returns a handle to the global
 * memory.
 */

HANDLE FAR PASCAL EndDialogHeader() /* end dialog header func*/
{
  LPSTR    lpCtrlData;  /* Long pointer to locked dialog
                           template memory  */

  lpCtrlData = GlobalLock( hDlgTemplate );
  if (lpCtrlData == NULL) {
    MessageBox ( GetFocus(),
                 (LPSTR)"Global lock",
                 (LPSTR)"Failed!", MB_OK);
    GlobalFree( hDlgTemplate );
  }

   /* 5th byte is address to # of items in dialog */
  lpCtrlData++;
  lpCtrlData++;
  lpCtrlData++;
  lpCtrlData++;

   /* set the number of items in control */
  *lpCtrlData = (BYTE)iItems;

  /* Unlock allocated memory */
  GlobalUnlock( hDlgTemplate );

  /* Return the handle to the Dialog Template */
  return (hDlgTemplate);
}

/* ------------- List of predefined dialog items ----------- */
/*
 *  This routine defines a list of predefined dialog items.
 *  This allows the user to just pass a number to get the
 *  attributes of a pre-defined control.
 *  You can add to this list or modify this to best fit
 *  your needs.
 *
 *  Warning: If you change one of the styles, it could effect
 *           other controls that expect the old behavior.
 */

void SetStyleClass(iStyle)
int  iStyle;
{
  /* if a iStyle is given
       then dtilControlClass will take on the default value. */

  switch ( iStyle ) {
     case DI0:      /* check box */
            DlgItem.dtilControlClass = BUTTONCLASS ;
            DlgItem.dtilStyle = BS_CHECKBOX | WS_TABSTOP |
                                WS_CHILD | WS_VISIBLE ;
            break;
     case DI1:      /* icon */
            DlgItem.dtilControlClass = STATICCLASS ;
            DlgItem.dtilStyle = SS_ICON | WS_BORDER |
                                WS_CHILD | WS_VISIBLE ;
            break;
     case DI2:      /* black box */
            DlgItem.dtilControlClass = STATICCLASS ;
            DlgItem.dtilStyle = SS_BLACKRECT | WS_CHILD |
                                WS_VISIBLE ;
            break;
     case DI3:      /* rectangle */
            DlgItem.dtilControlClass = STATICCLASS ;
            DlgItem.dtilStyle = SS_BLACKFRAME | WS_CHILD |
                                WS_VISIBLE ;
            break;
     case DI4:      /* left static text */
            DlgItem.dtilControlClass = STATICCLASS ;
            DlgItem.dtilStyle = SS_LEFT | WS_CHILD |
                                WS_VISIBLE ;
            break;
     case DI5:      /* multiline edit box */
            DlgItem.dtilControlClass = EDITCLASS ;
            DlgItem.dtilStyle = ES_LEFT | ES_MULTILINE |
                                ES_NOHIDESEL | ES_AUTOVSCROLL |
                                ES_AUTOHSCROLL |
                                WS_VSCROLL | WS_HSCROLL |
                                WS_BORDER | WS_TABSTOP |
                                WS_CHILD | WS_VISIBLE ;
            break;
     case DI6:      /* list box - sorted */
            DlgItem.dtilControlClass = LISTBOXCLASS ;
            DlgItem.dtilStyle = LBS_STANDARD | WS_CHILD |
                                WS_VISIBLE ;
            break;
     case DI7:      /* vertical scrollbar */
            DlgItem.dtilControlClass = SCROLLBARCLASS ;
            DlgItem.dtilStyle = SBS_VERT | WS_CHILD |
                                WS_VISIBLE ;
            break;
```

```
case DI8:      /* horizontal scrollbar */
        DlgItem.dtilControlClass = SCROLLBARCLASS ;
        DlgItem.dtilStyle = SBS_HORZ | WS_CHILD |
                            WS_VISIBLE ;
        break;
case DI9:      /* group box */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_GROUPBOX | WS_TABSTOP |
                            WS_CHILD | WS_VISIBLE ;
        break;
case DI10:     /* Push button */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_PUSHBUTTON | WS_TABSTOP |
                            WS_CHILD | WS_VISIBLE ;
        break;
case DI11:     /* radio button */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_RADIOBUTTON | WS_TABSTOP |
                            WS_CHILD | WS_VISIBLE ;
        break;
case DI12:     /* default push button */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_DEFPUSHBUTTON | WS_TABSTOP |
                            WS_CHILD | WS_VISIBLE ;
        break;
case DI13:     /* left check box */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_LEFTTEXT | BS_CHECKBOX |
                            WS_TABSTOP | WS_CHILD |
                            WS_VISIBLE ;
        break;
case DI14:     /* 3 auto state button */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_AUTO3STATE | WS_TABSTOP |
                            WS_CHILD | WS_VISIBLE ;
        break;
case DI15:     /* centered edit control */
        DlgItem.dtilControlClass = EDITCLASS ;
        DlgItem.dtilStyle = ES_CENTER | ES_MULTILINE |
                            WS_BORDER | WS_TABSTOP |
                            WS_CHILD  | WS_VISIBLE ;
        break;
case DI16:     /* right edit control */
        DlgItem.dtilControlClass = EDITCLASS ;
        DlgItem.dtilStyle = ES_RIGHT | ES_MULTILINE |
                            WS_BORDER | WS_TABSTOP |
                            WS_CHILD | WS_VISIBLE ;
        break;
```

```
case DI17:     /* left edit control */
        DlgItem.dtilControlClass = EDITCLASS ;
        DlgItem.dtilStyle = ES_LEFT  | WS_BORDER |
                            WS_TABSTOP |
                            WS_CHILD | WS_VISIBLE ;
        break;

case DI18:     /* listbox w/out sort */
        DlgItem.dtilControlClass = LISTBOXCLASS ;
        DlgItem.dtilStyle = LBS_NOTIFY | WS_BORDER |
                            WS_VSCROLL |
                            WS_CHILD | WS_VISIBLE ;
        break;
case DI19:     /* center static text */
        DlgItem.dtilControlClass = STATICCLASS ;
        DlgItem.dtilStyle = SS_CENTER | WS_BORDER |
                            WS_CHILD | WS_VISIBLE ;
        break;
case DI20:     /* right static text */
        DlgItem.dtilControlClass = STATICCLASS ;
        DlgItem.dtilStyle = SS_RIGHT | WS_CHILD |
                            WS_VISIBLE ;
        break;
default:       /* left edit control */
        DlgItem.dtilControlClass = STATICCLASS ;
        DlgItem.dtilStyle = SS_LEFT | WS_CHILD |
                            WS_VISIBLE ;
        break;

  } /* end switch */

}


WORD  lstrlen( lpszString )
LPSTR lpszString;     /* String to check */
{
   WORD Length;       /* Length of string */

   for ( Length = 0; *lpszString++ != '\0'; Length++ );

   return( Length );
}
```

Listing 6: *Das Listing des Beispielprogramms DlgTemp mit dynamischen Dialogboxen.*

Microsoft kündigt MS-DOS-Version 4.0 an:

Ein neues, freundliches DOS

Microsoft kündigte Ende Juli die Version 4.0 von MS-DOS an. Die neue Version zeichnet sich vor allem durch Funktionen aus, die es dem Anwender erlauben, größere Festplatten und Hauptspeicher-Erweiterungsboards zu nutzen, sowie eine visuelle Bedienungsoberfläche.

MS-DOS 4.0 bringt Verbesserungen in den Bereichen Bedienung, Verwaltung großer Festplatten-Dateien, Dateisystem und EMS-4.0-Unterstützung. Die Version 4.0 bietet neue Befehle wie MEM und eine Reihe von Verbesserungen bei alten Befehlen wie FORMAT, DELETE und GRAPHICS.

Die Verwendung der Versionsnummer 4.0 verursachte bei Kennern zunächst Verwunderung, denn eine MS-DOS Version 4.0 ist bereits seit langem auf dem Markt. Sie bietet einige Möglichkeiten für Multitasking in Netzwerken, ist jedoch nur von sehr wenigen Herstellern verwendet worden. Microsoft bezeichnet die jetzt angekündigte Version als »die offizielle Version 4.0«.

Die MS-DOS-Shell

Die für den Benutzer wichtigste Erweiterung von MS-DOS 4.0 ist die MS-DOS-Shell. Sie ist eine bildschirmorientierte, menügesteuerte Betriebssystem-Erweiterung für die Datei- bzw. Verzeichnisverwaltung und das Laden von Anwendungen. Sie kann über die Tastatur oder mit einer Maus bedient werden. Der speicherresidente Teil der MS-DOS-Shell belegt 4 Kbyte Hauptspeicher-Kapazität.

Eine DOS-Benutzerfläche von Microsoft gibt es schon seit einiger Zeit, sie wird zum Beispiel von Zenith unter dem Namen DOS-Manager angeboten. Die DOS-Shell hat damit nur noch wenig gemeinsam. Von der Darstellung und Handhabung her ist sie der OS/2-Presentation-Manager-Bedienungsoberfläche sehr ähnlich (*Bild 1*). Mit Hilfe der neuen DOS-Oberfläche kann der Anwender unter Zuhilfenahme von Pull-down-Menüs Anwendungsprogramme starten und DOS-Befehle aufrufen. Sie unterstützt den Anwender bei der Ausführung häufig benutzter Datei-Operationen, wie Kopieren und Löschen (*Bild 2*). Durch den Einsatz von Pull-down-Menüs und Dialogboxen braucht der Anwender sich nun die genauen Namen und die Syntax dieser Befehle nicht mehr unbedingt merken. Er kann auch seine Anwendungsprogramme direkt von der neuen Bedienungsoberfläche aus starten. Bei der Arbeit mit der DOS-Shell (wie auch dem Installationsprogramm) kann man jederzeit mit [F1] kontextsensitive Hilfe anfordern.

In der DOS-Shell können Anwendungsprogramme in das Hauptmenü (Start Programs) aufgenommen und so auf einfache Weise gestartet werden. Wenn viele Anwendungsprogramme aufgenommen werden, können sie auch gruppiert und in Untermenüs aufgeteilt werden (Bild 1).



Bild 1: *Das Hauptmenü der DOS-Shell ist erweiterbar und der Benutzer bekommt auf Tastendruck Hilfeinformationen.*

Die Verwaltung von Dateien findet in einem weiteren Programm, dem File System statt (Bild 2). Hier können in einem oder zwei Fenstern Dateien aus verschiedenen Verzeichnissen angezeigt und verwaltet werden. Wahlweise können auch alle Dateien in allen Verzeichnissen sortiert angezeigt werden. Dateien können mit Cursortasten oder der Maus einzeln oder in Gruppen ausgewählt werden. Die ausgewählten Dateien kann man dann kopieren, umbenennen, löschen, ansehen usw. Bei Beendigung des File Systems wird zum Hauptmenü (Start Programs) zurückverzweigt. Natürlich können auch unter der DOS-Shell wie bisher Befehlszeilen verwendet werden.

Die Möglichkeiten und die Benutzerschnittstelle der MS-DOS-4.0-Shell sind konsistent mit der Shell für das MS-OS/2-Betriebssystem Version 1.1, dem Presentation Manager. Microsoft hat überdies angekündigt, daß eine entsprechende Oberfläche in einer zukünftigen Version von Microsoft Windows verfügbar sein wird. Somit bieten diese Programme den Anwendern eine durchgängige Bedieneroberfläche von einfachen über mittlere bis zu Hochleistungs-PC.

Die MS-DOS-Shell ist jedoch ein MS-DOS-Anwendungsprogramm, also keine Programmierumgebung wie Microsoft Windows oder der MS-OS/2-Presentation Manager. Die DOS-Shell bietet deshalb auch kein Anwendungs-Programmier-Interface (API) zur Erstellung von Anwendungsprogrammen. Die MS-DOS-Shell ist darüber hinaus kein Ersatz für den COMMAND-Befehlsinterpreter (der wie bisher verwendet werden kann), sondern eine optionale Erweiterung.

Auch der SELECT-Befehl für die Installation von MS-DOS 4.0 arbeitet nun bildschirmorientiert und führt den Anwender Schritt für Schritt durch die Installationsprozedur, die notwendig ist, um MS-DOS 4.0 auf einer Festplatte oder einer Diskette zu installieren (*Bild 3*). SELECT erstellt während der Installation auch die entsprechenden CONFIG.SYS- und AUTOEXEC.BAT-Dateien.

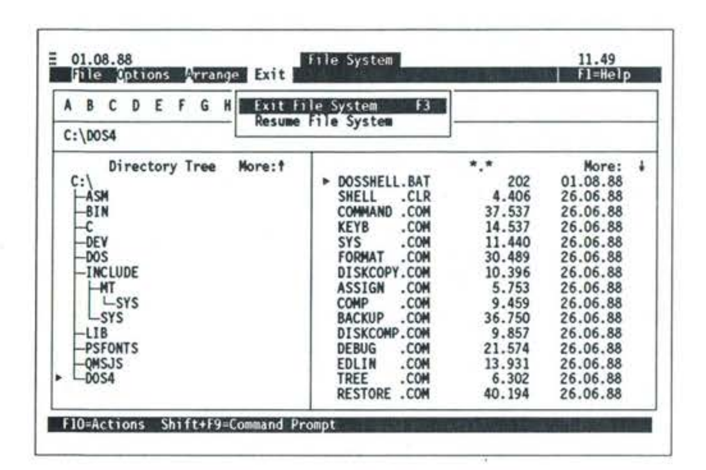


Bild 2: *Mit dem File-System der DOS-Shell können Dateien und Verzeichnisse verwaltet werden.*

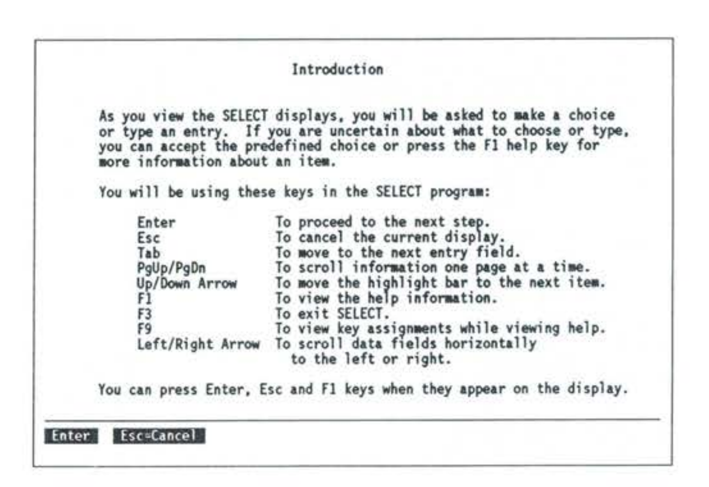


Bild 3: *Das Installationsprogramm SELECT arbeitet nun bildschirmorientiert.*

Viele Verbesserungen

Unter MS-DOS 4.0 kann der Anwender jetzt Dateien und Festplatten-Partitions erzeugen, die größer sind als 32 Mbyte. Die Version 4.0 ist jedoch noch in der Lage, auf alle Partitions und Dateien zuzugreifen, die unter früheren MS-DOS-Versionen erstellt worden sind. Der Anwender ist nun nicht mehr dazu gezwungen, große Dateien in mehrere kleine aufzuteilen, damit eine Speicherung auf der Festplatte möglich ist. Die Festplattendateien können unter MS-DOS 4.0 so groß sein wie die jeweilige Festplatten-Kapazität.

Bisherige Gerätetreiber konnten nur logische Sektornummern bis zu 16 Bit verarbeiten. Unter älteren DOS-Versionen war die Aufhebung der 32-Mbyte-Dateigrößе-Begrenzung nur durch 64-Kbyte-Sektoren möglich, wodurch dann aber sehr viel Plattenspeicher verschwendet wird. Die neuen, erweiterten Gerätetreiber sind in der Lage, logische Sektornummern von 32 Bit Länge zu verarbeiten. Die adressierbare maximale Festplattenkapazität beträgt deshalb nun 2 Gigabyte.

Besondere Leistungsverbesserungen konnten beim Datei-System erzielt werden. MS-DOS 4.0 verwendet beim Zugrif auf die FAT (Dateizuordnungstabelle) verbesserte Algorithmen, was sich vor allem beim wahlfreien Zugriff auf Dateien bemerkbar machen soll. Die Version 4.0 enthält auch eine erweiterte und verbesserte Unterstützung für das File-Caching.

Die FASTOPEN-Funktion ist nun in der Lage, die Informationen über geöffnete Dateien im Cache zwischenzuspeichern. Mit BUFFERS können mehr Datei-Puffer angelegt werden. Ein neuer »Caching-Algorithmus« vermindert dabei die Zugriffszeit auf Dateien, die in diesen Puffern gespeichert sind. Diese Leistungsverbesserungen machen sich besonders bei Anwendungen mit umfangreichen Plattenzugriffen bemerkbar, z.B. Datenbanken.

Neu ist auch die Unterstützung der »Expanded Memory Specification« (LIM/EMS 4.0), die von den Firmen Lotus, Intel, AST Research und Microsoft entwickelt wurde. Sie erlaubt Anwendungsprogrammen und Betriebssystem-Erweiterungen auf einen Hauptspeicher zuzugreifen, der mit Hilfe von Speichererweiterungskarten über 640 Kbyte hinaus erweitert wurde.

MS-DOS 4.0 kann Speichererweiterungen nach dem EMS-4.0-Standard zur Speicherung von DOS-Buffern und FASTOPEN-Daten verwenden. Je nach Systemkonfiguration können die Befehle FASTOPEN und BUFFERS zu einem erheblichen Bedarf an Speicherplatz im 640 Kbyte-Bereich führen, da jeder DOS-Puffer 512 Byte Speicher belegt. Diese Puffer können nun im Expanded Memory abgelegt werden, wodurch wieder mehr Speicher für Programme zur Verfügung steht.

Die Version 4.0 beinhaltet auch einen »LIMulator«, der als »Expanded Memory Manager« in einem System mit Extended Memory EMS-Speicher simuliert. Der LIMulator entspricht ebenfalls den EMS-4.0-Spezifikationen, läuft jedoch ausschließlich auf PCs mit 80386-Prozessoren.

Größerer Speicherbedarf

MS-DOS 4.0 setzt einen Hauptspeicher von mindestens 256 Kbyte Kapazität voraus. Der speicherresidente Teil von MS-DOS 4.0 ist 10 Kbyte größer als MS-DOS 3.3.

Da MS-DOS 4.0 konfigurierbar ist, hat der Anwender die Wahl, den darüber hinaus gehenden Speicherbedarf für das Betriebssystem festzulegen bzw. zu begrenzen. MS-DOS 4.0 erreicht oder übertrifft laut Microsoft alle wesentlichen Leistungsmerkmale der Version 3.3. Hinsichtlich des veröffentlichten API (Application Programming Interface) und der Utilities ist MS-DOS 4.0 kompatibel zu MS-DOS 3.3.

```
Address     Name          Size      Type
-------     ----          ----      ----
000000                    000400    Interrupt Vector
000400                    000100    ROM Communication Area
000500                    000200    DOS Communication Area

000700      IO            0025E0    System Program
                  CON                 System Device Driver
                  AUX                 System Device Driver
                  PRN                 System Device Driver
                  CLOCK$              System Device Driver
                  A: - D:             System Device Driver
                  COM1                System Device Driver
                  LPT1                System Device Driver
                  LPT2                System Device Driver
                  LPT3                System Device Driver
                  COM2                System Device Driver
                  COM3                System Device Driver
                  COM4                System Device Driver

002CE0      MSDOS         0088B0    System Program

00B590      IO            00F970    System Data
               PLUSDRV    000A60     DEVICE=
               ANSI       001180     DEVICE=
               MOUSE      002660     DEVICE=
               RAMDRIVE   0007A0     DEVICE=
               SMARTDRV   003B60     DEVICE=
               CACHE20    000180     DEVICE=
               JETSCRPT   001940     DEVICE=
                          000380     FILES=
                          000100     FCBS=
                          003E70     BUFFERS=
                          0008F0     LASTDRIVE=
                          000CD0     STACKS=
01AF10      COMMAND       001640    Program
01C560      IBMDOS        000030    -- Free --
01C5A0      COMMAND       000400    Environment
01C9B0      NDE           000090    Environment
01CA50      NDE           000D10    Program
01D770      MEM           000090    Environment
01D810      MEM           013220    Program
030A40      IBMDOS        06F5B0    -- Free --

  655360 bytes total memory
  655360 bytes available
  534496 largest executable program size

 1572864 bytes total extended memory
   36864 bytes available extended memory
```

Bild 4: *Der neue Befehl MEM gibt ausführliche Informationen über die Speicherbelegung.*

»Normale« Anwendungsprogramme laufen ohne Schwierigkeiten unter MS-DOS 4.0, Probleme treten jedoch bei einigen systemnahen Utitities oder Gerätetreibern auf. So laufen zum Beispiel die meisten Programme der Norton Utilities nicht mehr. Dies liegt zum einen natürlich an den Erweiterungen des Dateisystems, zum anderen aber auch daran, daß einige systemnahe Utilities undokumentierte Funktionsaufrufe, DOS-interne Dateistrukturen oder die Interrupte 25h und 26h für den direkten Plattenzugriff verwendet haben, die nun geändert wurden. So sollen auch Programme und Treiber von Microsoft wie Windows/386 und der bisherige CD-ROM-Treiber nicht unter MS-DOS 4.0 laufen. Von diesem Treiber wurde deshalb parallel eine neue erweiterte Version angekündigt.

MS-DOS 4.0 wurde mit der Ankündigung für Mikrocomputer-Hersteller freigegeben. Die ersten PCs mit der neuen MS-DOS-Version sollen voraussichtlich noch in diesem Quartal lieferbar sein. Laut Microsoft sind die Versionen 4.0 von MS-DOS und IBM-PC-DOS identisch.

Änderungen und Erweiterungen

Von den Änderungen und Erweiterungen blieben natürlich auch die DOS-Befehle nicht verschont, es handelt sich dabei jedoch so gut wie immer um Verbesserungen:

BUFFERS: Wenn Expanded Memory verwendet wird, können bis zu zu 10000 Puffer angegeben werden. Der Zugriff darauf wurde durch einen neuen Hash-Algorithmus verbessert. Auch können nun mehrere Sektoren gleichzeitig gelesen werden.

INSTALL: Mit diesem Befehl können speicherresidente Programme bereits bei der Konfiguration installiert werden.

REM: Es können jetzt auch in der Konfigurationsdatei mit REM Anmerkungen eingefügt werden.

COMMAND.COM: Der Befehlsprozessor erkennt nun auch Programmnamenserweiterungen. Wenn sich drei Dateien TEST.BAT, TEST.EXE und TEST.COM in einem Verzeichnis befinden, kann jede durch Eingabe mit der Erweiterung aufgerufen werden. Bisher wurde die Erweiterung ignoriert und die COM-Datei geladen.

DEL und ERASE: Mit Hilfe des Parameters /P (Prompt) kann man anfordern, daß die Namen der zu löschenden Dateien angezeigt und sie nur gelöscht werden, wenn man dies einzeln bestätigt. Wenn der Befehl DEL *.* verwendet wird, wird statt der nichtssagenden Meldung »Are you sure?« genau gesagt, daß alle Dateien im Verzeichnis gelöscht werden und gefragt, ob dies gewünscht wird.

FASTOPEN: Neben der Unterstützung von Extended Memory werden zusätzliche Informationen gespeichert, die die Verzeichnissuche beschleunigen und die Zugriffe auf die Dateizuordnungstabelle (FAT) verringern. Probleme, die sich bisher beim gleichzeitigen Einsatz von FASTOPEN und Plattenreorganisationsprogrammen (OPTIMIZE, SPEED DISK) ergeben haben, werden nun vermieden.

GRAPHICS: Das Hardcopy-Programm wurde wesentlich erweitert. So werden nun alle EGA- und VGA-Grafik-Bildschirmmodi unterstützt und es ist über eine ASCII-Datei (GRAPHICS.PRO) möglich, das Programm an verschiedene Drucker anzupassen.

MEM: Ein neuer Befehl, der Informationen über die Speicherbelegung, die im System befindlichen Treiber und speicherresidenten Programme ausgibt (*Bild 4*).

MODE: Erweiterungen zur Einstellung der Zeilen- und Spaltenzahl auf dem Bildschirm und Wiederholungsrate der Tastatur.

SYS: Das Betriebssystem kann nun auf jede Diskette/Platte übertragen werden, die genug Speicher und Verzeichniseinträge frei hat. SYS verschiebt Dateien und Ver-

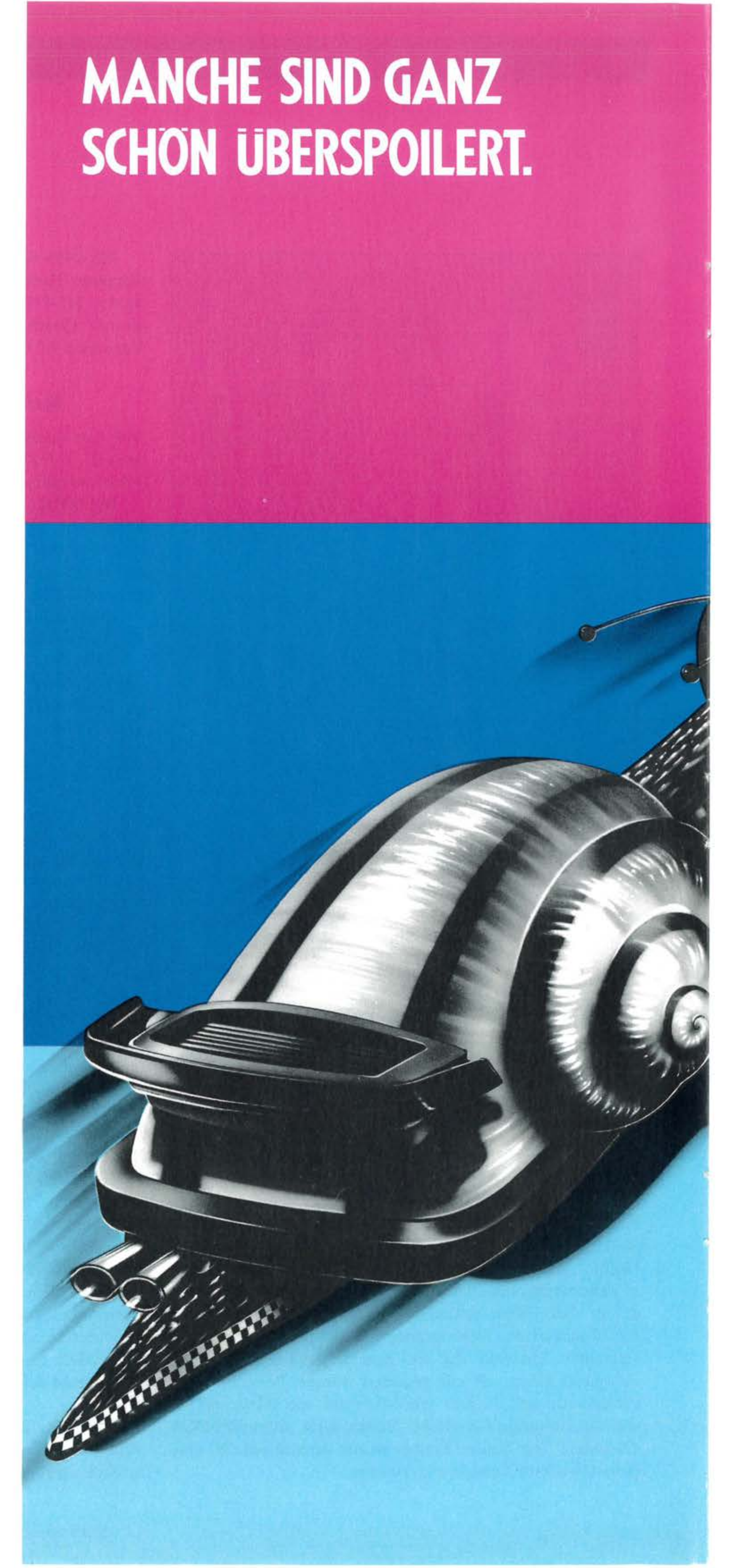
niseinträge wenn nötig, um die Systemdateien an die richtige Stelle zu positionieren.

TREE: Die Verzeichnisstruktur wird nun in eingerücktem Format und mit Liniengrafik angezeigt (*Bild 5*).

Es werden nun auch endlich die Bildschirmadapter EGA, VGA und MCGA unterstützt. Das bedeutet, das zum Beispiel der Bildschirmtreiber ANSI.SYS oder Programme wie MORE, EDLIN und DIR /P erkennen, wenn auf den neuen Bildschirmadaptern mit mehr als 25 Zeilen gearbeitet wird. Auch die neuen Bildschirmmodi können über ANSI-Escapesequenzen eingestellt werden.

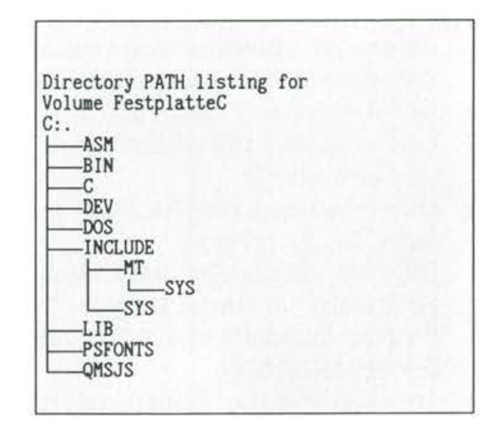
Unterm Strich ist MS-DOS 4.0 eine sinnvolle Weiterentwicklung, deren hervorstechendstes Merkmal die DOS-Shell ist. Sie wird vor allem Benutzern, die nur gelegentlich mit dem PC arbeiten oder sich nicht auf der Bit-Ebene mit der Materie beschäftigen wollen, eine willkommene Erleichterung sein.

(Fortsetzung nächste Seite)



Die Probleme, die bei einigen systemnahen Programmen wegen der Erweiterungen des Dateisystems auftreten, werden durch die Vorteile (größere Dateien, schnellerer Zugriff) mehr als ausgeglichen.

Günter Jürgensmeier



```
Directory PATH listing for
Volume FestplatteC
C:.
├──ASM
├──BIN
├──C
├──DEV
├──DOS
├──INCLUDE
│   ├──MT
│   │  └──SYS
│   └──SYS
├──LIB
├──PSFONTS
└──QMSJS
```

Bild 5: *Die Ausgabe von TREE ist übersichtlicher geworden.*

Das erste umfassende Tabellenwerk für PCs:

Die PC-Referenz für Programmierer

In fast jedem Fachgebiet gibt es sie, und sie werden auch stark genutzt: Es gibt Tabellenwerke für die Chemie, Physik, Mathematik, Maschinenbau und so weiter, nur im Computerbereich gab es so etwas bisher noch nicht. Das hat sich mit dem PC-Sourcebook von Thom Hogan (deutscher Titel: Die PC-Referenz für Programmierer) nun geändert. In diesem Buch sind die wesentlichen Fakten über Hard- und Systemsoftware des PC-Bereichs in Tabellenform gesammelt. Das Buch läßt sich nur schwer beschreiben, man muß es gesehen haben. Deshalb haben wir auf den folgenden Seiten einige Beispiele abgedruckt. Dies ist natürlich nur ein kleiner Ausschnitt aus der Informationsfülle der über 500 Seiten. Die wichtigsten Bereiche sind:

- Allgemeine Tabellen (numerische Konvertierungen, Datenformate, Zeichensatztabellen)
- DOS-Übersichten (DOS-Befehle und ihre Parameter, DOS-Utilities, Diskettenstruktur, Dateiaufbau, Speicherverwendung)
- DOS-Funktionen (Int 21h, FCB- & PSP-Aufbau, Fehlercodes, Gerätetreiber)
- BIOS-Funktionen (Int 10h inklusive EGA- und VGA-Funktionen, Int 11h-Int 1Ah)
- Weitere Interrupte (Int 24h - Int2Fh, Maus Int 33h, LIM/EMS Int 67h)
- Microsoft Windows (Eingabegeräte, Dateien, Datenformate, Windows-Funktionen)
- PC-Hardware (Tastatur, Bildschirmadapter, Schnittstellen, Uhr)
- Chips und Register (CPUs, Koprozessoren, Peripherie-Chips, Jumper, Controller)
- Hardwarebeschreibungen (Steckerbelegungen)

Softwareentwicklern wird damit eine Informationsquelle geboten, die viele andere Bücher überflüssig macht. Fast alle Informationen lassen sich nun in einem Werk finden.

Ein interessanter Hinweis noch zum Schluß: Dieses Buch ist ein ausgezeichnetes Beispiel für die Qualitäten von Microsoft Excel. Alle Tabellen wurde damit erfaßt und auf einem Postscript-Drucker ausgegeben. Excel ist also nicht nur für Kalkulationstabellen geeignet, sondern ganz allgemein für jede Art von Tabelle, auch wenn darin nicht gerechnet wird. Besonders wenn man eine Tabelle in Desktop Publishing-Qualität ausgedruckt haben möchte, ist Microsoft Excel derzeit unschlagbar.

Thom Hogan: »The programmer's PC sourcebook: charts and tables for the IBM PC, compatibles, and the MS-DOS Operating System, including the new IBM Personal System/2 computers«, Redmond: Microsoft Press, 1988; 525 Seiten A4; ISBN 1-55615-118-7; $24.95.

Die deutsche Version wird im Oktober '88 erscheinen:

Thom Hogan: »Die PC-Referenz für Programmierer«, München: Microsoft Press im Systhema Verlag, 1988; ca. 525 Seiten A4; ISBN 3-89390-250-3; ca. DM 69,-.

PC- und Windows-Zeichensatztabelle

Auf der Doppelseite in der Mitte dieses Heftes finden Sie eine Übersicht, in der die Zeichen des Standard-PC-Zeichensatzes und ihre Codes dargestellt sind. Diese Tabelle soll Programmierern bei Codewandlungen und Umrechnungen behilflich sein und Anwendern die Eingabe von Zeichen erleichtern, die nur über die [Alt]-Taste erreichbar sind; ein solches Zeichen kann eingegeben werden, indem die [Alt]-Taste gedrückt wird und dann (mit gedrückter [Alt]-Taste) der dezimale Wert dieses Zeichens auf der Zehnertastatur eingegeben wird.

Im einzelnen werden in der Tabelle für jedes Zeichen folgende Angaben gemacht:

1. Der dezimale Wert (oben links); er ist mit einem D gekennzeichnet.
2. Das Zeichen aus dem Windows-Zeichensatz (oben rechts). Bei Werten unter 32 werden hier die Controlcodes angegeben (^A), da diese Zeichen unter Windows nicht belegt sind.
3. Der oktale Wert (Mitte links, gekennzeichnet durch O); besonders für C-Programmierer wichtig.
4. Der hexadezimale Wert (Mitte rechts, H); hilfreich beim Programmieren in Assembler, bei der Arbeit mit Debuggern und ganz allgemein bei der systemnahen Programmierung.
5. Der binäre Wert (unten, B); für systemnahe Programmierung (Bit-Verknüpfungen usw.).

In der ersten schmalen Spalte bei den Werten unter 32 sind die ASCII-Namen der Steuerzeichen aufgeführt.

Mit Hilfe dieser Tabelle läßt sich nun leicht feststellen, welches Zeichen welchem dezimalen, oktalen, hexadezimalen oder binären Wert entspricht.

Da es bei einigen Programmen, aber auch einigen Druckern, immer noch zu Problemen mit den Umlauten kommt, hier eine Tabelle mit dem Paragraphenzeichen, Umlauten und scharfem S und den Codes im DIN- und IBM-Zeichensatz:

	DIN	IBM
§	64	21
Ä	91	142
Ö	92	153
Ü	93	154
ä	123	132
ö	124	148
ü	125	129
ß	126	225

Günter Jürgensmeier

BILDSCHIRMSPEICHERAUFBAU UND AUSGABEWERTE

		MDA	CGA	EGA	VGA
Speicher	*Pufferadresse*	B0000	B8000	*	*
	Puffergröße	4 K	16 K	64 K - 256 K	256 K
	Seiten im Puffer	1	4 bis 8	Max. 8	Max. 8
	Verwendete I/O-Ports	3B0-3BF	3D0-3DF	3B0-3DF	3B0-3DF
Ausgabe	*Bandbreite*	16.257 MHz**	14.30 MHz	14.3 bis 16.3 MHz	28 MHz
	Horiz. Abtastrate	18.432 KHz**	15.75 KHz	15.7 bis 21.8 KHz	31.5 KHz
	Vert. Abtastrate	50 Hz**	60 Hz	60 Hz	50 bis 70 Hz
	Max. horiz. Pixel	720	640	640	720
	Max. vert. Pixel	350	200	350	480
	Größe der Zeichenmatrix	9x14	8x8	9x14 oder 8x8	9x16
	Wirkliche Zeichengröße	7x9	7x7 oder 5x7	7x9 oder 7x7	7x9
System	*Zugriff auf CPU*	nicht beim Refresh	jederzeit	jederzeit	jederzeit
	Datenübertragungsrate	1.8 M/sek	1.5 M/sek	?	?
Ausstattung	*Lichtstift*	Nein	Ja	Ja	Nein
	Fernseheranschluß	Nein	Ja	Nein	Nein
	Digitaler RGB-Anschluß	Nein	Ja	Ja	Nein
	Analoger RGB-Anschluß	Nein	Nein	Nein	Ja
	Direkter Video-Anschluß	Ja	Ja	Ja	Nein
	Farbpalette	Keine	16 Farben	64 Farben	256 K Farben
	Feature-Anschluß	Nein	Nein	Ja	Nein
	Modulator-Anschluß	Nein	Ja	Nein	Nein

Hinweise: * B0000 bei 32 K oder B8000 bei 32 K oder A0000 bei 64 K oder A0000 bei 128 K.
Bei der EGA-Karte wird ein16 K BIOS-Erweiterungsmodul in den Prozessorspeicher bei Adresse C0000 gelegt.
** Zusammen mit dem monochromen Bildschirm von IBM.

ZEICHENATTRIBUTE DES MONOCHROMEN BILDSCHIRMADAPTERS (MDA)

Bitnummer

7	6	5	4	3	2	1	0	Funktion	erlaubte Werte
X								Blinken	0=nicht blinken, 1=blinken
	X	X	X					Hintergrund	000=schwarzer Hintergrund 111=weißer Hintergrund
				X				Intensität	0=normal, 1=erhöhte Intensität
					X	X	X	Vordergrund	000 = schwarzes Zeichen 001 = unterstreichen 111 = weißes Zeichen

ZEICHENATTRIBUTE VON CGA, EGA UND VGA

Bitnummer

7	6	5	4	3	2	1	0	Funktion	erlaubte Werte	
X								Blinken	0=nicht blinken, 1=blinken	
	X	X	X					Hintergrund	000=schwarz 001=blau 010=grün 011=kobaltblau 100=rot 101=violett 110=braun 111=grau	
				X				Intensität	0=normal, 1=höhere Intensität	
					X	X	X	Vordergrund	000=schwarz 001=blau 010=grün 011=kobaltblau 100=rot 101=violett 110=braun 111=grau	mit Intensität: dunkelgrau mit Intensität: hellblau mit Intensität: hellgrün mit Intensität: hellkobaltblau mit Intensität: hellrot mit Intensität: hellviolett mit Intensität: gelb mit Intensität: weiß

Hinweis: · Unsichtbare Zeichen erhält man, wenn man Zeichen auf einen Hintergrund der gleichen Farbe stellt. (z.B. weiß auf weiß).

ÜBERSICHT ÜBER DIE BILDSCHIRMMODI

Einzeheiten der BIOS-Modi						*Unterstützt von Adapter*				
Modus	*Art*	*Zeilen*	*Spalten*	*Auflösung*	*Farben*	*MDA*	*CGA*	*EGA*	*MCGA*	*VGA*
0	Text	25	40	320x200	16		X	X	X	X
1	Text	25	40	320x200	16		X	X	X	X
2	Text	25	80	640x200	16		X	X	X	X
3	Text	25	80	640x200	16		X	X	X	X
4	Grafik	25	40	320x200	4		X	X	X	X
5	Grafik	25	40	320x300	4		X	X	X	X
6	Grafik	25	80	640x200	2		X	X	X	X
7	Text	25	80	720x350*	Mono	X		X		X
13	Grafik	25	40	320x200	16			X		X
14	Grafik	25	80	640x200	16			X		X
15	Grafik	25	80	640x350	Mono			X		X
16	Grafik	25	80	640x350	16			X		X
17	Grafik	30	80	640x480	2				X	X
18	Grafik	30	80	640x480	16					X
19	Grafik	25	40	320x200	256				X	X

Hinweise:
- Die EGA-Werte gehen von 256K RAM aus.
- Die Modi 8 bis 12 werden nur vom PCJr verwendet.

* 720x400 bei VGA

HÄUFIGE STRINGFORMATE

DOS-Befehlszeile

Länge	String
0	1 … 127

DOS-Stringanzeige (Int 21, Funktion 9)

String	$
0	Länge+1

ASCIIZ

String	0
0	Länge+1

*C**

String	0
0	Länge+1

*Pascal**

StrLänge	String
0	2 … Länge+2

*BASICA** (Microsoft BASIC)*

Länge	Zeiger
0	1 … 2

-------->

String
Zeiger … Zeiger + Länge

Hinweise:

* Nicht alle C- und Pascal-Compiler verwenden exakt diese Formate, doch dies ist der jeweils anerkannte Standard.

** In BASICA werden der String und die Informationen darüber im Speicher nicht hintereinander gespeichert.

WERTEBEREICHE VERSCHIEDENER 8087-DATENFORMATE

	kleinster möglicher Wert	*größter möglicher Wert*
8087 Word Integer	-32,768	32,767
8087 Short Integer	-2,147,483,648	2,147,483,647
8087 Long Integer	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
8087 Packed Decimal	-(10^18)-1	(10^18)-1
8087 Short Real	8.43 x 10^-37	3.37 x 10^38
8087 Long Real	4.19 x 10^-307	1.67 x 10^308
8087 Temporary Real	3.4 x 10^-4932	1.2 x 10^4932
IEEE Floating Point	4.19 x 10^-307	1.67 x 10^308

VERZEICHNISEINTRÄGE

Offset	*Länge*	*Beschreibung*	*Format*	*Anmerkungen*
0 (0)	8 Byte	Dateiname**	ASCII-Zeichen	mit Leerzeichen auf Feldlänge aufgefüllt
8 (8)	3 Byte	Dateityp (Erweiterung)	ASCII-Zeichen	mit Leerzeichen auf Feldlänge aufgefüllt
B (11)	Byte	Dateiattributbyte	Bit codes: Bit 0 = nur Lesen Bit 1 = versteckt Bit 2 = System Bit 3 = Datenträgername Bit 4 = Verzeichnis Bit 5 = Archivierung Bit 6 = nicht verwendet Bit 7 = nicht verwendet	s. unten
C (12)	10 Byte	RESERVIERT		
16 (22)	Wort	Zeit letzter Aktualisierung*	kodiertes Wort: (unsigned 16 Bit) Zeit = Std*2048+Min*32+Sek/2	
18 (24)	Wort	Datum letzter Aktualisierung*	kodiertes Wort: (unsigned 16 Bit) Datum = (JJ-1980)*512+MM*32+TT	
1A (26)	Wort	Nummer des Startclusters*	Wort, binäre Ganzzahl*	
1C (28)	Doppelwort	Dateigröße*	Doppelwort, binäre Ganzzahl*	

Hinweise: · Dateiname und Erweiterung werden nicht durch einen Punkt getrennt.

* Niederwertiges Byte zuerst

** Das erste Byte des Dateinamens kennzeichnet den Status eines Verzeichniseintrags:

00H = Eintrag/Name wurde nie verwendet
05H = das erste Zeichen des Namens ist eigentlich E5H (Sigma)
E5H = Eintrag wurde verwendet, Datei wurde jedoch gelöscht
2EH = Eintrag ist ein Verzeichnis (wenn das zweite Byte auch 2EH ist, enthält das Clusterfeld die Clusternummer des übergeordneten Verzeichnis)

DATEIATTRIBUTBYTE

Bitnummer

7	*6*	*5*	*4*	*3*	*2*	*1*	*0*	*Bedeutung wenn 1*	*Bedeutung wenn 0*
							X	Datei nur lesbar	Datei les- und beschreibbar
						X		versteckte Datei	sichtbare Datei
					X			Systemdatei	normale Datei
				X				Datenträgername	normale Datei
			X					Verzeichnisname	normale Datei
		X						Datei seit letzter Sicherung geändert	Datei seit letzter Sicherung nicht geändert
X	X							RESERVIERT	RESERVIERT

Version: DOS 1.x verwendet nur die Bits 0-3

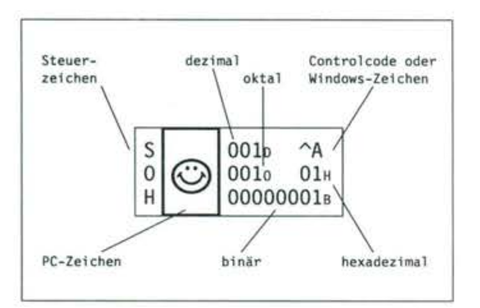
Hinweise: · Die Bits 3 und 4 schließen sich aus; es kann keines, das eine oder das andere gesetzt sein, doch nicht beide.

· Nur bei einer Datei (im Stammverzeichnis) darf Bit 3 gesetzt sein.

TYPISCHE REGISTERVERWENDUNG UNTER DOS

Register	*Standardverwendung*	*Bits*	*Anmerkungen*
AX	allgemeines Datenregister	16	übergibt MS-DOS-Parameter, gibt Fehler zurück
AH	Funktionsnummer beim Aufruf	8	enthält Funktionsnummer beim DOS-Aufruf (INT 21H)
AL	Fehlerrückgabe	8	Fehlernummer, wenn Carry gesetzt ist
BX	Basisregister Datensegment	16	gibt auch Daten zurück (z.B. Handle)
CX	Schleifenzähler	16	wird gelegentlich für Datenübergabe verwendet
DX	allgemeines Datenregister	16	wird oft als als Zeiger auf Daten verwendet (Offset zu DS)
SP	Stapelzeiger	16	
IP	Befehlszeiger	16	
BP	Basisregister Stapelsegment	16	
DS	Datensegment	16	wird normalerweise mit DX verwendet
SS	Stapelsegment	16	wird normalerweise mit BX oder CX verwendet
SI	Zeiger auf Quelle bei Stringoperationen	16	
DI	Zeiger auf Ziel bei Stringoperationen	16	
Flags	Carryflag=1=Fehler; Carryflag=0=kein Fehler	1	wird hauptsächlich von DOS 2.1 und später verwendet

Microsoft SYSTEM JOURNAL

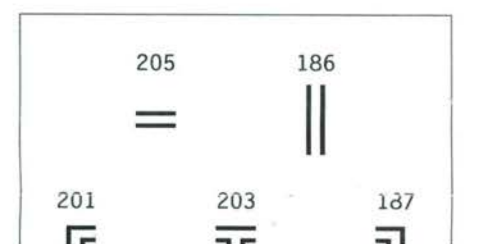


196 ─ 179 │

218 ┌ 194 ┬ 191 ┐

195 ├ 197 ┼ 180 ┤

192 └ 193 ┴ 217 ┘



Steuerzeichen	PC-Zeichen	Dezimal	Controlcode	Oktal	Hex	Binär
NUL		000D	^@	000O	00H	00000000B
SOH	☺	001D	^A	001O	01H	00000001B
STX	☻	002D	^B	002O	02H	00000010B
ETX	♥	003D	^C	003O	03H	00000011B
EOT	♦	004D	^D	004O	04H	00000100B
ENQ	♣	005D	^E	005O	05H	00000101B
ACK	♠	006D	^F	006O	06H	00000110B
BEL	•	007D	^G	007O	07H	00000111B
BS	◘	008D	^H	010O	08H	00001000B
HT	○	009D	^I	011O	09H	00001001B
LF	◙	010D	^J	012O	0AH	00001010B
VT	♂	011D	^K	013O	0BH	00001011B
FF	♀	012D	^L	014O	0CH	00001100B
CR	♪	013D	^M	015O	0DH	00001101B
SO	♫	014D	^N	016O	0EH	00001110B
SI	☼	015D	^O	017O	0FH	00001111B
DL	►	016D	^P	020O	10H	

PC-Zeichen	Dezimal	Windows-Zeichen	Oktal	Hex	Binär
	032D		040O	20H	00100000B
!	033D	!	041O	21H	00100001B
"	034D	"	042O	22H	00100010B
#	035D	#	043O	23H	00100011B
$	036D	$	044O	24H	00100100B
%	037D	%	045O	25H	00100101B
&	038D	&	046O	26H	00100110B
'	039D	'	047O	27H	00100111B
(	040D	(	050O	28H	00101000B
)	041D	)	051O	29H	00101001B
*	042D	*	052O	2AH	00101010B
+	043D	+	053O	2BH	00101011B
,	044D	,	054O	2CH	00101100B
-	045D	-	055O	2DH	00101101B
.	046D	.	056O	2EH	00101110B
/	047D	/	057O	2FH	00101111B
0	048D	0	060O	30H	

PC-Zeichen	Dezimal	Windows-Zeichen	Oktal	Hex	Binär
@	064D	@	100O	40H	01000000B
A	065D	A	101O	41H	01000001B
B	066D	B	102O	42H	01000010B
C	067D	C	103O	43H	01000011B
D	068D	D	104O	44H	01000100B
E	069D	E	105O	45H	01000101B
F	070D	F	106O	46H	01000110B
G	071D	G	107O	47H	01000111B
H	072D	H	110O	48H	01001000B
I	073D	I	111O	49H	01001001B
J	074D	J	112O	4AH	01001010B
K	075D	K	113O	4BH	01001011B
L	076D	L	114O	4CH	01001100B
M	077D	M	115O	4DH	01001101B
N	078D	N	116O	4EH	01001110B
O	079D	O	117O	4FH	01001111B
P	080D	P	120O	50H	

PC-Zeichen	Dezimal	Windows-Zeichen	Oktal	Hex	Binär
`	096D	`	140O	60H	01100000B
a	097D	a	141O	61H	01100001B
b	098D	b	142O	62H	01100010B
c	099D	c	143O	63H	01100011B
d	100D	d	144O	64H	01100100B
e	101D	e	145O	65H	01100101B
f	102D	f	146O	66H	01100110B
g	103D	g	147O	67H	01100111B
h	104D	h	150O	68H	01101000B
i	105D	i	151O	69H	01101001B
j	106D	j	152O	6AH	01101010B
k	107D	k	153O	6BH	01101011B
l	108D	l	154O	6CH	01101100B
m	109D	m	155O	6DH	01101101B
n	110D	n	156O	6EH	01101110B
o	111D	o	157O	6FH	01101111B
p	112D	p	160O	70H	

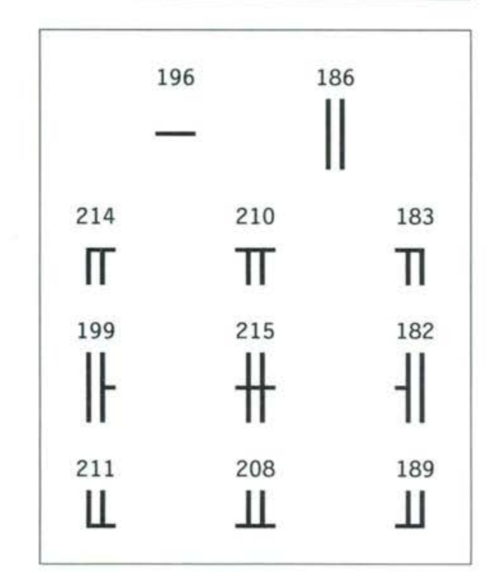
PC-Zeichen	Dezimal	Windows-Zeichen	Oktal	Hex	Binär
É	144D	■	220O	90H	10010000B
æ	145D	'	221O	91H	10010001B
Æ	146D	'	222O	92H	10010010B
ô	147D	■	223O	93H	10010011B
ö	148D	■	224O	94H	10010100B
ò	149D	■	225O	95H	10010101B
û	150D	■	226O	96H	10010110B
ù	151D	■	227O	97H	10010111B
ÿ	152D	■	230O	98H	10011000B
Ö	153D	■	231O	99H	10011001B
Ü	154D	■	232O	9AH	10011010B
¢	155D	■	233O	9BH	10011011B
£	156D	■	234O	9CH	10011100B
¥	157D	■	235O	9DH	10011101B
Pt	158D	■	236O	9EH	10011110B
ƒ	159D	■	237O	9FH	10011111B
Ç	128D	■	200O	80H	

PC-Zeichen	Dezimal	Windows-Zeichen	Oktal	Hex	Binär
░	176D	°	260O	B0H	10110000B
▒	177D	±	261O	B1H	10110001B
▓	178D	²	262O	B2H	10110010B
│	179D	³	263O	B3H	10110011B
┤	180D	´	264O	B4H	10110100B
╡	181D	µ	265O	B5H	10110101B
╢	182D	¶	266O	B6H	10110110B
╖	183D	·	267O	B7H	10110111B
╕	184D	¸	270O	B8H	10111000B
╣	185D	¹	271O	B9H	10111001B
║	186D	º	272O	BAH	10111010B
╗	187D	»	273O	BBH	10111011B
╝	188D	¼	274O	BCH	10111100B
╜	189D	½	275O	BDH	10111101B
╛	190D	¾	276O	BEH	10111110B
┐	191D	¿	277O	BFH	10111111B
á	160D		240O	A0H	

PC-Zeichen	Dezimal	Windows-Zeichen	Oktal	Hex	Binär
╨	208D	Ð	320O	D0H	11010000B
╤	209D	Ñ	321O	D1H	11010001B
╥	210D	Ò	322O	D2H	11010010B
╙	211D	Ó	323O	D3H	11010011B
╘	212D	Ô	324O	D4H	11010100B
╒	213D	Õ	325O	D5H	11010101B
╓	214D	Ö	326O	D6H	11010110B
╫	215D	■	327O	D7H	11010111B
╪	216D	Ø	330O	D8H	11011000B
┘	217D	Ù	331O	D9H	11011001B
┌	218D	Ú	332O	DAH	11011010B
█	219D	Û	333O	DBH	11011011B
▄	220D	Ü	334O	DCH	11011100B
▌	221D	Ý	335O	DDH	11011101B
▐	222D	Þ	336O	DEH	11011110B
▀	223D	ß	337O	DFH	11011111B
└	192D	À	300O	C0H	

PC-Zeichen	Dezimal	Windows-Zeichen	Oktal	Hex	Binär
≡	240D	ð	360O	F0H	11110000B
±	241D	ñ	361O	F1H	11110001B
≥	242D	ò	362O	F2H	11110010B
≤	243D	ó	363O	F3H	11110011B
⌠	244D	ô	364O	F4H	11110100B
⌡	245D	õ	365O	F5H	11110101B
÷	246D	ö	366O	F6H	11110110B
≈	247D	■	367O	F7H	11110111B
°	248D	ø	370O	F8H	11111000B
●	249D	ù	371O	F9H	11111001B
•	250D	ú	372O	FAH	11111010B
√	251D	û	373O	FBH	11111011B
ⁿ	252D	ü	374O	FCH	11111100B
²	253D	ý	375O	FDH	11111101B
■	254D	þ	376O	FEH	11111110B
	255D	ÿ	377O	FFH	11111111B
α	224D	à	340O	E0H	

201	203	187
╔	╦	╗
204	206	185
╠	╬	╣
200	202	188
╚	╩	╝



205	179	
═	│	
213	209	184
╒	╤	╕
198	216	181
╞	╪	╡
212	207	190
╘	╧	╛

Ctrl	Glyph	Dec	Char	Oct	Hex	Bin
DLE	►	016D	^P	020O	10H	00010000B
DC1	◄	017D	^Q	021O	11H	00010001B
DC2	↕	018D	^R	022O	12H	00010010B
DC3	‼	019D	^S	023O	13H	00010011B
DC4	¶	020D	^T	024O	14H	00010100B
NAK	§	021D	^U	025O	15H	00010101B
SYN	▬	022D	^V	026O	16H	00010110B
ETB	↨	023D	^W	027O	17H	00010111B
CAN	↑	024D	^X	030O	18H	00011000B
EM	↓	025D	^Y	031O	19H	00011001B
SUB	→	026D	^Z	032O	1AH	00011010B
ESC	←	027D	^[	033O	1BH	00011011B
FS	∟	028D	^\	034O	1CH	00011100B
GS	↔	029D	^]	035O	1DH	00011101B
RS	▲	030D	^^	036O	1EH	00011110B
US	▼	031D	^_	037O	1FH	00011111B

Glyph	Dec	Char	Oct	Hex	Bin
0	048D	0	060O	30H	00110000B
1	049D	1	061O	31H	00110001B
2	050D	2	062O	32H	00110010B
3	051D	3	063O	33H	00110011B
4	052D	4	064O	34H	00110100B
5	053D	5	065O	35H	00110101B
6	054D	6	066O	36H	00110110B
7	055D	7	067O	37H	00110111B
8	056D	8	070O	38H	00111000B
9	057D	9	071O	39H	00111001B
:	058D	:	072O	3AH	00111010B
;	059D	;	073O	3BH	00111011B
<	060D	<	074O	3CH	00111100B
=	061D	=	075O	3DH	00111101B
>	062D	>	076O	3EH	00111110B
?	063D	?	077O	3FH	00111111B

Glyph	Dec	Char	Oct	Hex	Bin
P	080D	P	120O	50H	01010000B
Q	081D	Q	121O	51H	01010001B
R	082D	R	122O	52H	01010010B
S	083D	S	123O	53H	01010011B
T	084D	T	124O	54H	01010100B
U	085D	U	125O	55H	01010101B
V	086D	V	126O	56H	01010110B
W	087D	W	127O	57H	01010111B
X	088D	X	130O	58H	01011000B
Y	089D	Y	131O	59H	01011001B
Z	090D	Z	132O	5AH	01011010B
[	091D	[	133O	5BH	01011011B
\	092D	\	134O	5CH	01011100B
]	093D	]	135O	5DH	01011101B
^	094D	^	136O	5EH	01011110B
_	095D	_	137O	5FH	01011111B

Glyph	Dec	Char	Oct	Hex	Bin
p	112D	p	160O	70H	01110000B
q	113D	q	161O	71H	01110001B
r	114D	r	162O	72H	01110010B
s	115D	s	163O	73H	01110011B
t	116D	t	164O	74H	01110100B
u	117D	u	165O	75H	01110101B
v	118D	v	166O	76H	01110110B
w	119D	w	167O	77H	01110111B
x	120D	x	170O	78H	01111000B
y	121D	y	171O	79H	01111001B
z	122D	z	172O	7AH	01111010B
{	123D	{	173O	7BH	01111011B
		124D			174O	7CH	01111100B
}	125D	}	175O	7DH	01111101B
~	126D	~	176O	7EH	01111110B
⌂	127D	■	177O	7FH	01111111B

Glyph	Dec	Char	Oct	Hex	Bin
Ç	128D	■	200O	80H	10000000B
ü	129D	■	201O	81H	10000001B
é	130D	■	202O	82H	10000010B
â	131D	■	203O	83H	10000011B
ä	132D	■	204O	84H	10000100B
à	133D	■	205O	85H	10000101B
å	134D	■	206O	86H	10000110B
ç	135D	■	207O	87H	10000111B
ê	136D	■	210O	88H	10001000B
ë	137D	■	211O	89H	10001001B
è	138D	■	212O	8AH	10001010B
ï	139D	■	213O	8BH	10001011B
î	140D	■	214O	8CH	10001100B
ì	141D	■	215O	8DH	10001101B
Ä	142D	■	216O	8EH	10001110B
Å	143D	■	217O	8FH	10001111B

Glyph	Dec	Char	Oct	Hex	Bin
á	160D		240O	A0H	10100000B
í	161D	¡	241O	A1H	10100001B
ó	162D	¢	242O	A2H	10100010B
ú	163D	£	243O	A3H	10100011B
ñ	164D	¤	244O	A4H	10100100B
Ñ	165D	¥	245O	A5H	10100101B
ª	166D	¦	246O	A6H	10100110B
º	167D	§	247O	A7H	10100111B
¿	168D	¨	250O	A8H	10101000B
⌐	169D	©	251O	A9H	10101001B
¬	170D	ª	252O	AAH	10101010B
½	171D	«	253O	ABH	10101011B
¼	172D	¬	254O	ACH	10101100B
¡	173D	-	255O	ADH	10101101B
«	174D	®	256O	AEH	10101110B
»	175D	¯	257O	AFH	10101111B

Glyph	Dec	Char	Oct	Hex	Bin
└	192D	À	300O	C0H	11000000B
┴	193D	Á	301O	C1H	11000001B
┬	194D	Â	302O	C2H	11000010B
├	195D	Ã	303O	C3H	11000011B
─	196D	Ä	304O	C4H	11000100B
┼	197D	Å	305O	C5H	11000101B
╞	198D	Æ	306O	C6H	11000110B
╟	199D	Ç	307O	C7H	11000111B
╚	200D	È	310O	C8H	11001000B
╔	201D	É	311O	C9H	11001001B
╩	202D	Ê	312O	CAH	11001010B
╦	203D	Ë	313O	CBH	11001011B
╠	204D	Ì	314O	CCH	11001100B
═	205D	Í	315O	CDH	11001101B
╬	206D	Î	316O	CEH	11001110B
╧	207D	Ï	317O	CFH	11001111B

Glyph	Dec	Char	Oct	Hex	Bin
α	224D	à	340O	E0H	11100000B
β	225D	á	341O	E1H	11100001B
Γ	226D	â	342O	E2H	11100010B
π	227D	ã	343O	E3H	11100011B
Σ	228D	ä	344O	E4H	11100100B
σ	229D	å	345O	E5H	11100101B
µ	230D	æ	346O	E6H	11100110B
τ	231D	ç	347O	E7H	11100111B
Φ	232D	è	350O	E8H	11101000B
Θ	233D	é	351O	E9H	11101001B
Ω	234D	ê	352O	EAH	11101010B
δ	235D	ë	353O	EBH	11101011B
∞	236D	ì	354O	ECH	11101100B
φ	237D	í	355O	EDH	11101101B
∈	238D	î	356O	EEH	11101110B
∩	239D	ï	357O	EFH	11101111B

DIE SPEICHERVERWENDUNG DES BIOS — *Bitnummern*

Adresse	*Länge*	*Beschreibung*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	*Anmerkungen*
40:00	Wort	Basisadresse COM1									
40:02	Wort	Basisadresse COM2									
40:04	Wort	Basisadresse COM3									wird nur vom PS/2 BIOS unterstützt
40:06	Wort	Basisadresse COM4									wird nur vom PS/2 BIOS unterstützt
40:08	Wort	Basisadresse LPT1									
40:0A	Wort	Basisadresse LPT2									
40:0C	Wort	Basisadresse LPT3									
40:0E	Wort	Basisadresse LPT4									nur PC, XT, AT & Convertible (AP)
40:10	Byte	Installierte Hardware 1	X	X							Anzahl Floppy-Laufwerke (beginnt mit 0)
					X	X					Videomodus (01=40x25 Farbe, 10=80x25 Farbe, 11=80x25 mono)
							X	X			RESERVIERT (alte PC und PCJr: Bits 2-3 geben inst. Speicher an)
								X			Zeigegerät installiert (nur PS/2)
									X		Num. Koprozessor installiert (nicht beim PCjr oder Convertible)
										X	Floppylaufwerke installiert
40:11	Byte	Installierte Hardware 2	X	X							Anzahl Druckeradapter
					X						Internes Modem (nur Convertible)
						X					Joystick installiert
							X	X	X		Anzahl RS-232-Adapter
										X	RESERVIERT (PCJr=DMA-Gerät installiert)
40:12	Byte	Status Selbstest beim Start									nur Convertible
40:13	Wort	Speichergröße									in KByte (0 bis 640)
40:15	Wort	RESERVIERT									
40:17	Byte	Tastatursteuerung 1	X								Einfügemodus aktiv
				X							CapsLock-Modus aktiv
					X						NumLock-Modus aktiv
						X					ScrollLock-Modus aktiv
							X				Alt-Taste gedrückt
								X			Ctrl-Taste gedrückt
									X		Linke Shift-Taste gedrückt
										X	Rechte Shift-Taste gedrückt
40:18	Byte	Tastatursteuerung 2	X								Taste Einfügen gedrückt
				X							Taste Caps-Lock gedrückt
					X						Taste Num-Lock gedrückt
						X					Taste Scroll-Lock gedrückt
							X				Pausenmodus aktiv
								X			Taste Systemanfrage gedrückt
									X		Linke Alt-Taste gedrückt
										X	Rechte Alt-Taste gedrückt
40:19	Byte	Alternative Tastatureingabe									
40:1A	Wort	Zeiger auf Start des Tastaturpuffers									zeigt auf das erste Zeichen im Tastaturpuffer
40:1C	Wort	Zeiger auf Ende des Tastaturpuffers									zeigt auf das letzte Zeichen im Tastaturpuffer
40:1E	32 Byte	Tastaturpuffer									
40:3E	Byte	Status Nachjustierung der Floppy	X								Interruptflag
				X	X	X					RESERVIERT
							X				Laufwerk 3 nachjustieren
								X			Laufwerk 2 nachjustieren
									X		Laufwerk 1 nachjustieren
										X	Laufwerk 0 nachjustieren
40:3F	Byte	Motorstatus der Floppy	X								Schreib-/Leseoperation
				X							RESERVIERT
					X	X					Ausgewähltes Laufwerk (Bitwert = Laufwerk)
							X				Status Laufwerk 3 Motor AN
								X			Status Laufwerk 2 Motor AN
									X		Status Laufwerk 1 Motor AN
										X	Status Laufwerk 0 Motor AN
40:40	Byte	Zähler Motor aus									
40:41	Byte	Status der vorigen Floppy-Operation	X								Laufwerk nicht bereit
				X							Fehlerhafte Positionierung
					X						Allgemeiner Controllerfehler
						X					CRC-Fehler beim Lesen der Diskette
							X				DMA-Überlauf bei der Operation
								X			Geforderter Sektor nicht gefunden
									X		Adreßmarke nicht gefunden
										X	Ungültige Laufwerkparameter
											Kein Fehler

DIE SPEICHERVERWENDUNG DES BIOS *Bitnummern*

Adresse	*Länge*	*Beschreibung*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	*Anmerkungen*
40:41									X	X	Schreibschutzfehler
								X	X		Diskette gewechselt
							X			X	DMA über 64 K Segmentgrenze
							X	X			Mediatyp nicht gefunden
40:42	7 Byte	Statusbyte Floppycontroller									
40:49	Byte	Bildschirmmodus									
40:4A	Wort	Anzahl Spalten									
40:4C	Wort	Länge des Bildschirmpuffers in Byte									
40:4E	Wort	Adresse des Bildschirmpuffers									
40:50	Wort	Cursorposition Seite 1									
40:52	Wort	Cursorposition Seite 2									
40:54	Wort	Cursorposition Seite 3									
40:56	Wort	Cursorposition Seite 4									
40:58	Wort	Cursorposition Seite 5									
40:5A	Wort	Cursorposition Seite 6									
40:5C	Wort	Cursorposition Seite 7									
40:5E	Wort	Cursorposition Seite 8									
40:60	Wort	Cursortyp									
40:62	Byte	Aktuelle Bildschirmseite									
40:63	Wort	Basisadresse Bildschirmcontroller									
40:65	Byte	Aktuelle Einstellung 3x8-Register									
40:66	Byte	Aktuelle Einstellung 3x9-Register									
40:67	Doppelwort	Zeiger auf Resetroutine									nur PS/2 (außer Model 30)
40:6B	Byte	RESERVIERT									
40:6C	Doppelwort	Timerzähler									
40:70	Byte	Timer Überlaufflag									<>0: Timer über 24 Std.
40:71	Byte	Status Breaktaste									
40:72	Wort	Resetflag									1234H=kein Speichertest; 4321H=Speicherinhalt erhalten (PS/2) 5678H=System angehalten (Conv.); 9ABCH=Herst.test (Conv.) ABCDH=Systemwarteschleife (nur Convertible)
40:74	Byte	Status vorige Festplattenoperation									(bei PS/2 nicht verwendet)
40:75	Byte	Anzahl Festplatten									
40:76	Byte	Laufwerkssteuerung Festplatte									nur XT
40:77	Byte	Controllerport Festplatte									nur XT
40:78	Byte	Zeitüberschreitungswert Drucker 1									
40:79	Byte	Zeitüberschreitungswert Drucker 2									
40:7A	Byte	Zeitüberschreitungswert Drucker 3									
40:7B	Byte	Zeitüberschreitungswert Drucker 4									nur PC, XT und AT
40:7C	Byte	Zeitüberschreitungswert COM1									
40:7D	Byte	Zeitüberschreitungswert COM2									
40:7E	Byte	Zeitüberschreitungswert COM3									
40:7F	Byte	Zeitüberschreitungswert COM4									
40:80	Wort	Offsetpointer Beginn Tastaturpuffer									
40:82	Wort	Offsetpointer Ende Tastaturpuffer									
40:84	Byte	Bildschirmzeilen (-1)									
40:85	Wort	Zeichenhöhe (Bytes/Zeichen)									
40:87	Byte	Bildchirmsteuerstatus 1									
40:88	Byte	Bildschirmsteuerstatus 2									
40:89	Wort	RESERVIERT									
40:8B	Byte	Mediasteuerung	X	X							Letzte Floppy-Datenrate * (AT, XT nach 10.1.85, PS/2)
					X	X					Letzter Floppy-Schrittwert (AT, XT nach 10.1.85, PS/2)
							X	X	X	X	RESERVIERT
40:8C	Byte	Status Festplattencontroller									nur AT, XT nach 10.1.85 und PS/2
40:8D	Byte	Fehlerstatus Festplattencontroller									nur AT, XT nach 10.1.85 und PS/2
40:8E	Byte	Interruptsteuerung Festplatte									nur AT, XT nach 10.1.85 und PS/2
40:8F	Byte	RESERVIERT									
40:90	Byte	Mediastatus Laufwerk 0	X	X							Laufwerk-Datenrate* (AT, XT nach 10.1.85, PS/2)
					X						Doppelstepping notwendig (AT, XT nach 10.1.85, PS/2)
						X					Medium eingerichtet (AT, XT nach 10.1.85, PS/2)
							X				RESERVIERT
								X	X	X	Laufwerk-/Mediumstatus** (AT, XT nach 1.10.85, PS/2)
40:91	Byte	Mediastatus Laufwerk 1	X	X							Laufwerk-Datenrate* (AT, XT nach 10.1.85, PS/2)
					X						Doppelstepping notwendig (AT, XT nach 10.1.85, PS/2)
						X					Medium eingerichtet (AT, XT nach 10.1.85, PS/2)
							X				RESERVIERT

DIE SPEICHERVERWENDUNG DES BIOS *Bitnummern*

Adresse	*Länge*	*Beschreibung*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	*Anmerkungen*
								X	X	X	Laufwerk-/Mediumstatus** (AT, XT nach 1.10.85, PS/2)
40:92	Wort	RESERVIERT									
40:94	Byte	Aktueller Zylinder Laufwerk 0									
40:95	Byte	Aktueller Zylinder Laufwerk 1									
40:96	Byte	Modusstaus Tastatur	X								ID wird gelesen
				X							Letztes Zeichen war erstes ID-Zeichen
					X						Num-Lock erzwingen wenn ID gelesen und KBX
						X					Tastatur mit 101/102 Tasten installiert
							X				rechte Alt-Taste gedrückt
								X			rechte Ctrl-Taste gedrückt
									X		Letzter Code war E0, versteckter Code
										X	Letzter Code war E1, versteckte Code
40:97	Byte	LED-Flags Tastatur	X								Fehlerflag Tastaturübertragung
				X							Aktualisierung Modusanzeige
					X						Empfangsflag zurücksenden
						X					Empfangsbestätigung erhalten
							X				RESVERVIERT (muß 0 sein)
								X	X	X	Statusbit LED
40:98	Wort	Offset auf Warteroutine									
40:9A	Wort	Segment auf Warteroutine									
40:9C	Wort	Wartezähler (niederwertiges Wort)									in Mikrosekunden
40:9E	Wort	Wartezähler (höherwertiges Wort)									in Mikrosekunden
40:A0	Byte	Flag warten aktiv	X								verstrichene Wartezeit
				X	X	X	X	X	X		RESERVIERT
										X	Int 15H Funktion 86H (Wait) aufgetreten
40:A1	7 Byte	RESERVIERT									
40:A8	Doppelwort	Zeiger auf Bildschirmparametertab.									nur EGA und PS/2
40:AC	Doppelwort	Dynamischer Sicherungsbereich Ptr									nur EGA und PS/2
40:B0	Doppelwort	Zeiger zus. Zeichengenerator (Text)									nur EGA und PS/2
40:B4	Doppelwort	Zeiger zus. Zeichengenerator (Grafik)									nur EGA und PS/2
40:B8	Doppelwort	Zweiter Sicherungszeiger									nur PS/2 (nicht Modell 30)
40:BC	8 Byte	RESERVIERT									Wird nur auf Null gesetzt
40:C0	64 Byte	RESERVIERT									
50:00	Byte	Statusbyte PrtSc									

Version: Das erweiterte BIOS von PS/2 verwendet Speicher im oberen Bereich als erweiterten BIOS-Datenbereich.
Hinweis: * Laufwerksdaten: 00 = 500 KB/Sek., 01 = 300 KB/Sek., 10 = 250 KB/Sek., 11 = RESERVIERT
** Werte des Laufwerksmediastatus derzeit im IBM-BIOS-Handbuch nicht dokumentiert

AUFBAU DES PROGRAMMSEGMENTPRÄFIX

Offset	*Länge*	*typischer Inhalt*	*Beschreibung*	*Anmerkungen*
0 (0)	Wort	CD20H	Int 20H Beendigungsadresse	
2 (2)	Wort		Ende des zugewiesenen Speicherblocks	eine Segmentadresse
4 (4)	Byte	00H	RESERVIERT	
5 (5)	5 Byte		CALL FAR zum DOS-Funktionsverteiler	veraltet
A (10)	Doppelwort		Int 22H Adresse der Beendigungsroutine	bei Prgende wiederhergest., Form: IP:CS
E (14)	Doppelwort		Int 23H Adresse der Ctrl-Break-Routine	gespeichert in der Form IP:CS
12 (18)	Doppelwort		Int 24H Adresse Routine für krit. Fehler	gespeichert in der Form IP:CS
16 (22)	Wort		PSP des übergeodneten Prozesses	eine Segmentadresse
18 (24)	20 Byte	FF=verfügbar	Handle-Tabelle	1 Byte pro Handle, bit 7=nicht übernommen
2C (44)	Wort		Blockadresse der Umgebung	eine Segmentadresse
2E (46)	Doppelwort		RESERVIERT	
32 (50)	Wort	14H,00H	Größe der Handle-Tabelle	ab DOS 3.3 größere Tabellen möglich
34 (52)	Doppelwort	12H,00H	Adresse Handle-Tabelle (wenn nicht 18H)	ab DOS 3.3 Tabellenadresse möglich
38 (56)	23 Byte		RESERVIERT	
50 (80)	Wort	CD21H	Int 21H DOS-Aufruf	
52 (82)	Byte	CBH	RET FAR	
53 (83)	9 Byte		RESERVIERT	
5C (92)	36 Byte		Ungeöffneter Standard-FCB 1	
6C (108)	20 Byte		Ungeöffneter Standard-FCB 2	überlagert FCB 1
80 (128)	Byte		Länge der Befehlszeilenparameter	gleichzeitig Begin der Standard-DTA
81 (129)	127 Byte		Befehlszeilenparameter	beginnt mit Leerzeichen, endet mit CR

Hinweis: Die einzigen sinnvollen Informationen in den FCBs sind Laufwerksnummer und Dateiname.

Still going strong:

Geschichte und Aufbau von MS-DOS

Nach acht Jahren und zahlreichen Versionen ist MS-DOS in der Version 4.0 nun so komfortabel und umfangreich wie nie zuvor. Alterserscheinungen sind ihm noch nicht anzusehen und es wird ihm immer noch eine glänzende Zukunft vorausgesagt. Anlaß genug, um uns einmal ausführlich mit der Entwicklungsgeschichte und dem Aufbau von MS-DOS zu beschäftigen.

Angefangen hatte alles eigentlich schon im Oktober 1980. Damals kamen die Manager von IBM auf Microsoft zu und fragten, ob sie ihnen nicht ein Betriebssystem für ihren in der Entwicklung befindlichen 16-Bit-Mikrocomputer programmieren könnten. Vorher hatten Sie bei Digital Research nachgefragt, dem Entwickler des damals wichtigsten Mikrocomputer-Betriebssystems CP/M. Der Geschäftsführer von Digital Research, Gary Kildall, war bei einem geplanten Treffen jedoch verhindert, die IBM-Manager fühlten sich vor den Kopf gestoßen und gingen zu Microsoft, die damals vor allem wegen ihres BASIC-Interpreters bekannt waren.

Geschäftsführer von Microsoft war damals, genau wie heute, William H. Gates III, oder kurz Bill Gates, der zusammen mit seinem Partner Paul Allen Microsoft BASIC entwickelt und erfolgreich gemacht hatte. Bill Gates, der eine in der Mikrocomputerszene seltene Mischung aus profundem Computerfachwissen und gutem Geschäftssinn besitzt, witterte die Chance seines Lebens und sagte ohne zu zögern zu. Er besaß zwar zu diesem Zeitpunkt kein Betriebssystem, doch war das für ihn kein Hinderungsgrund. Er kannte eine kleine Firma mit dem Namen Seattle Computer Products, deren Mitarbeiter Tim Paterson ein einfaches 16-Bit-Betriebssystem für den Intel 8086 entwickelt hatte. Paterson nannte das Betriebssystem ursprünglich QDOS, was für »Quick'n Dirty Operating System«, also etwa »Das schnell und unsauber programmierte Betriebssystem«, stand. Später taufte er es jedoch auf den seriöseren Namen 86-DOS um. Bill Gates kaufte nun kurzerhand für 50000 Dollar die Rechte an diesem Betriebssystem und machte es zur Grundlage von MS-DOS. Im November 1980 unterschrieb IBM den Entwicklungsvertrag mit Microsoft, im Februar 1981 hatte Microsoft einen Prototypen von MS-DOS auf einem Prototypen des IBM-PC zum Laufen gebracht. In den folgenden sechs Monaten bis zur Ankündigung des IBM-PC wurde noch weiter daran gefeilt, so daß es stabil und für den Büroalltag geeignet war.

Die Versionen

Tim Paterson, der ab Mai 1981 bei Microsoft beschäftigt war, hatte 86-DOS auch unter dem Gesichtspunkt entwikkelt, daß die Umsetzung von CP/M-Programmen, die zumeist in 8080-Assembler geschrieben waren, einfach sein sollte.

Version	Datum	Beschreibung
86-DOS	1980	Originalversion von Seattle Computer Products
PC-DOS 1.0	August 1981	PC-DOS für den Original-PC, einseitige Disketten
PC-DOS 1.1	Mai 1982	doppelseitige Disketten
MS-DOS 1.25	1982	erste OEM-Version
MS-DOS & PC-DOS 2.0	März 1983	PC XT mit Festplatte, hierarchisches Dateisystem
PC-DOS 2.1	Oktober 1983	für IBM PCjr und tragbaren PC
MS-DOS 2.1	1983	Unterstützung internationaler Belange
MS-DOS & PC-DOS 3.0	August 1984	für PC AT mit High-Density-Diskettenlaufwerken und größeren Festplatten
MS-DOS & PC-DOS 3.1	März 1985	Netzwerkunterstützung
MS-DOS & PC-DOS 3.2	1986	Unterstützung von 3 1/2-Zoll-Disketten
MS-DOS & PC-DOS 3.3	April 1987	PS/2-Unterstützung, Aufteilung von Festplatten mit über 32 Mbyte.
MS-DOS & PC-DOS 4.0	Juli 1988	bildschirmorientierte Benutzerschnittstelle, EMS-Unterstützung

Tabelle 1: *Die wichtigsten Versionen von MS-DOS.*

Deshalb wurden die meisten Funktionen und Strukturen (z.B. Dateisteuerblöcke - FCBs) übernommen, damit CP/M-Programme von einem einfachen Umsetzungsprogramm in 8086-Code umgesetzt werden konnten. So war die erste Version von MS-DOS eigentlich nicht viel mehr als ein CP/M für 8086-Prozessoren.

MS-DOS ist jedoch im Laufe der Zeit mehrmals geändert und stark erweitert worden (es existieren sogar mehrere Versionen, die nie offiziell zu kaufen waren). Auch wenn es zum Teil um Fehlerkorrekturen ging, waren neue Versionen meistens mit Hardwareankündigungen von IBM verbunden. In Tabelle 1 sind die wichtigsten DOS-Versionen aufgeführt.

Jede DOS-Version wurde durch die notwendigen Erweiterungen und Verbesserungen natürlich auch größer.

Die Version 1.0 kam noch mit 16 Kbyte Speicher aus, was auch notwendig war, da der erste IBM-PC nur mit 64 Kbyte Speicher ausgestattet war. Die Version 2 benötigte bereits 24 Kbyte und für ernsthafte Programme war deshalb schon ein Speicherausbau von 128 Kbyte Voraussetzung, 256 Kbyte waren jetzt ein Traumausbau. In der Version 3 benötigt DOS 36 Kbyte und oft sogar mehr, wenn weitere Gerätetreiber oder Netzwerkunterstützung geladen werden. Ein Standard-PC mit weniger als 512 Kbyte wurde nun fast schon belächelt. Mit der Version 4.0 hat sich der Speicherbedarf noch einmal um 10 Kbyte erhöht, sie verbessert die Speichersituation jedoch insofern, daß sie die Verwendung von Speicher nach dem LIM/EMS-Standard für Puffer erlaubt, so daß der Programmspeicher von MS-DOS, der auf PCs auf 640 Kbyte beschränkt ist, wieder ein wenig entlastet wird.

Die Versionen im Detail

1.0: Die Originalversion für IBM-PCs, in den wesentlichen Teilen dem damaligen Betriebssystemstandard für 8-Bit-Computer, CP/M, sehr ähnlich. Zu den Verbesserungen gegenüber CP/M gehörte vor allem eine verbessertes Dateisystem, das den Dateizugriff beschleunigte und es dem Benutzer ermöglichte, auch ohne umständliche Anmeldung (mount oder Control-C bei anderen Systemen) Disketten zu wechseln. Auch können beim Start automatisch Befehle ausgeführt werden, die in einer Datei mit dem Namen AUTOEXEC.BAT stehen. Da auf 8086-Computern Programme größer als 64 Kbyte sein können (die Grenze bei 8-Bit-Computern), wurde mit EXE-Dateien ein neues Programmformat angeboten, in dem auch größere Programme relokatierbar (positionsunabhängig) gespeichert werden können.

Laut Bill Gates entsprach das Betriebssystem in dieser Version noch nicht seinen Vorstellungen: »Im Grunde wollten wir eigentlich eines schreiben, das mehr wie die Version 2 aussehen sollte, mit hierarchischem Dateisystem und allem ...«. Doch die Entwicklungszeit reichte dafür nicht, so daß sich Microsoft auf eine lauffähige Untermenge beschränken mußte. Insgesamt bestand die Version 1.0 aus 4000 Zeilen Assemblercode; der speicherresidente Teil war im lauffähigen Zustand 8 Kbyte groß. Der Betriebssystemkern bestand schon damals aus drei Programmen:

`IBMBIO.COM` bzw. `IO.SYS`, die Schnittstelle zum BIOS mit Zeichen-Ein-/Ausgabe und Diskettenzugriffen.
`IBMDOS.COM` bzw. `MSDOS.SYS`, der eigentliche Kern des Betriebssystems mit Programmierschnittstelle.
`COMMAND.COM`, der Befehlsprozessor, oder wie es heute schöner heißt, die »Benutzerschnittstelle«, also der Teil von DOS, mit dem es der Benutzer meistens zu tun hat, die von vielen wenig geschätzte »Eingabeaufforderung«.

Die Version 1.0 wurde nur von IBM angeboten und allgemein PC-DOS genannt. Microsoft hatte jedoch schon vorgesorgt und eine hardwareunabhängige interne Struktur entwickelt, die die Anpassung auf andere System leicht machen sollte.

Die Version 1.0 unterstützte nur einseitige Disketten mit 160 Kbyte Speicherkapazität, an Festplatten war zu dieser Zeit wegen der hohen Preise noch nicht zu denken.

1.1/1.25: In dieser Version wurde MS-DOS nun auch von anderen Herstellern wie Compaq und Zenith angeboten. Bei IBM war die Versionsmnummer 1.1, bei Microsoft und seinen OEMs 1.25. Es wurden einige Fehler ausgebessert, die Geräteunabhängigkeit verbessert und doppelseitige Disketten mit 320 Kbyte unterstützt. In Deutschland war der wichtigste Computer, auf dem diese Version lief, der von vielen immer noch geschätzte Sirius 1, der jedoch von Anfang an auf einer 5 1/4-Zoll-Diskette 600 Kbyte und später sogar 1,2 Mbyte speicherte. Dem Erfolg des Sirius 1 und der engen Zusammenarbeit seiner Entwicklungsmannschaft mit Microsoft ist es zu verdanken, daß MS-DOS auch in Deutschland schnell der Standard wurde, denn der IBM-PC wurde zu diesem Zeitpunkt noch nicht in Deutschland verkauft.

2.0: Die Version 2.0 war der gewaltigste Schritt in der Entwicklung von MS-DOS und brachte es in die Form, die ursprünglich geplant war. Sie brachte die Unterstützung von Festplatten, wodurch auch ein neues Dateisystem notwendig wurde. Die wichtigste Erweiterung dieser Version war deshalb eine hierarchische Verzeichnisstruktur, die von Unix bzw. Xenix übernommen wurde. Um dem PC als offenes System gerecht zu werden, konnten nun beim Start Gerätetreiber installiert werden, die sogar eingebaute Treiber ersetzen durften. Doch wurden auch noch viele andere Erweiterungen vorgenommen, die DOS endlich zu einem »richtigen« Betriebssystem machten. Im einzelnen brachte die Version 2.0:

- ein hierarchisches Dateisystem;
- Dateizugriff über Handles (Dateinummern) statt FCBs (Dateisteuerblöcke), die keinen Platz für Pfadnamen haben;
- Ein-/Ausgabeumleitung, Pipes und Filter;
- Hintergrunddruck mit PRINT.COM;
- Datenträgernamen (volume label);
- Erweiterung der Dateiattibute;
- die Konfigurationsdatei CONFIG.SYS;
- installierbare Geträtetreiber um Herstellern von Peripheriegeräten die Einbindung ihrer Systeme zu erleichtern;
- ANSI-Bildschirmtreiber;
- Anpaßbarkeit an andere Sprachen und länderspezifische Funktionen, z.B. zur Abfrage von Datums- und Zeitformat;

- Verwaltung einer Programmumgebung (environment) durch DOS;
- ein austauschbarer Befehlsprozessor;
- Disketten mit 9 Sektoren und 360 Kbyte.

Die Version 2.0 kam im März 1983 auf den Markt und war immer noch das Produkt einer sehr kleinen Entwicklungsmannschaft von sechs Leuten, die sich zum Teil im Betriebssystem verewigt haben. So heißt einer der Leiter der MS-DOS-Entwicklungsmannschaft Mark Zbikowski, dessen Initialen »MZ« als Kennzeichnung am Anfang jeder EXE-Datei und am Anfang jedes von MS-DOS zugewiesenen Speicherblocks steht.

Ab der Version 2.01, die nicht von IBM verkauft wurde, unterstützte MS-DOS auch das japanische Kanji-Alphabet mit seinen über 7000 Schriftzeichen.

Trotz seiner vielen Erweiterungen wurden nur 24 Kbyte Speicher benötigt. Doch mit dem IBM XT, seiner Festplatte und dem neuen DOS begann eine Entwicklung, bei der immer leistungsfähigere Programme immer mehr Speicher, intern und extern, benötigten, so daß die Standardspeichergröße bald bei 256 Kbyte lag.

2.1: In dieser Version wurden nur geringfügige Änderungen vorgenommen. Vor allem ging es dabei um die Unterstützung des in Deutschland nie verkauften PCjr und den tragbaren PC von IBM. Für seine eigenen OEMs verknüpfte Microsoft die Versionen 2.01 und 2.1 zu einer Version 2.11, die zur ersten Standardversion vieler Hersteller wurde und auch heute noch wegen seiner Kompaktheit und Robustheit von einigen Anbietern mitgeliefert wird oder bereits im ROM eingebaut ist. 2.11 wurde in über 10 Sprachen übersetzt, darunter chinesisches Kanji und koreanisch (Hanguel-Zeichensatz). Gleich nachdem diese Version fertiggestellt war, begann Microsoft mit der Arbeit an der Version 3.0.

3.0: Die Entwicklung der Version 3.0 erwies sich jedoch als schwierig, da inzwischen viele OEMs unterstützt werden mußten, der Einsatz in Netzwerkumgebungen geplant war und Microsoft bei dem stetigen Wachstum von MS-DOS darauf achten mußte, den Programmcode übersichtlich und wartbar zu halten. Die Entwicklung beanspruchte mehr Zeit, als geplant, so daß die Version 3.0 im August 1984 von IBM, zusammen mit dem AT, ohne Netzwerkunterstützung angeboten wurde. Einige andere OEMs boten dieses DOS unter der Versionsnummer 3.05 an.

In der Version 3.0 kam die Unterstützung des IBM-AT mit Disketten höherer Kapazität (1,2 Mbyte) und weitere Festplattentypen hinzu. Weitere wichtige Erweiterungen:

- Steuerung des Spoolers (PRINT.COM) durch Anwendungsprogramme;
- erweiterte Fehlerhinweise und Fehlerbehebungstips für Programme;
- schnellerer Dateizugriff;
- Dateischutz auf Datei- und Datensatzebene.

3.1: Die Version wurde kurz nach der Version 3.0 herausgebracht. Sie ist lange Zeit bei vielen Herstellern der Standard gewesen und ist es zum Teil heute noch. Es wurden nun Platten in Netzwerken und der gemeinsame Zugriff auf Dateien unterstützt. Auch wurden einige Fehler ausgemerzt und die internen Dateifunktionen auf Handles umgestellt. Deshalb wurde der neue Konfigurationsbefehls FCBS eingeführt, der die Kompatibilität mit Programmen sicherstellen sollte, die noch über FCBs auf Dateien zugreifen.

3.2: Wichtigste Neuerung dieser Version war die Unterstützung von 3 1/2-Zoll-Laufwerken. Durch Einschluß solcher Programme wie DISKCOMP in MS-DOS und Umschreiben einiger anderer externen Utilities wurde die volle Kompatibilität der DOS-Versionen anderer Hersteller mit der von IBM sichergestellt.

3.3: Es kommen zwei neue Befehle hinzu (NLSFUNC und FASTOPEN), die Unterstützung internationaler Zeichensätze wird verbessert und die PS/2-Reihe von IBM unterstützt.

4.0: Die Version 4.0 erlaubt größere Festplatten (bis 2 Gigabyte) und die Nutzung von Hauptspeichererweiterungen nach dem LIM/EMS-Standard 4.0. Die für den Benutzer wichtigste Erweiterung von MS-DOS 4.0 ist die MS-DOS-Shell, eine bildschirmorientierte, menügesteuerte Betriebssystem-Erweiterung für die Datei- bzw. Verzeichnisverwaltung und das Laden von Anwendungen. Damit ist nun eine Art der Bedienung zum Standard gemacht worden, die im PC-Bereich vor allem durch Microsoft Windows populär gemacht worden ist.

Die Entwicklung von MS-DOS dürfte damit jedoch noch nicht abgeschlossen sein, es wird sicherlich auch in Zukunft mit weiteren Versionen zu rechnen sein. Jeder Benutzer hat sicherlich gelegentlich die eine oder andere Funktion vermißt. Dies hat zu einem großen Erfolg solcher Programme wie die Norton Utilities oder PC-Tools geführt. Solche Möglichkeiten werden jedoch wahrscheinlich nicht Bestandteil von DOS werden. Doch kann ich mir gut vorstellen, daß es einmal eine Hilfefunktion ähnlich wie unter OS/2 mit dem Befehl HELP geben wird, der einem Hinweise zu Fehlermeldungen gibt.

MS-DOS bzw. PC-DOS soll nach Angaben von Bill Gates inzwischen weltweit auf über 10 Millionen PCs laufen und es sollen über 20000 Anwendungsprogramme dafür geschrieben worden sein. Das ist mit Abstand die größte Softwaresammlung für eine Computerumgebung und eine Basis, die eine Garantie für den Bestand von MS-DOS ist. MS-DOS ist mit acht Jahren für die Verhältnisse der schnellebigen Computerszene zwar bereits uralt, doch wird es vielen von uns auch in den kommenden Jahren noch ein treuer Begleiter sein. Und ganz gleich, wie lange es noch verwendet werden wird, einen Ehrenplatz in der Mikrocomputer-Geschichte hat es sich schon lange verdient.

Günter Jürgensmeier

Mitteilungen .... Mitteilungen .... Mitteilungen

Verbesserung der Windows-Oberfläche geplant

Mit der Vorstellung der grafischen Bedienungsoberfläche von PC-DOS 4.0 bzw. MS-DOS 4.0 vor wenigen Wochen wurde zugleich die Oberflächenstruktur des OS/2 Presentation Managers vorgestellt, die mit derjenigen von PC-DOS 4.0 bzw. MS-DOS 4.0 identisch sein wird. Das Erscheinungsbild des Presentation Managers war bislang nur MS-OS/2-Entwicklern bekannt.

Gegenüber dem Vorbild Microsoft Windows weisen DOS 4.0 und der Presentation Manager einige Neuerungen auf. So erleichtert eine grafische File- bzw. Pfadstruktur in Bauform die Übersicht über Filesysteme auf Datenträgern. Dateien werden im Grafikmodus sowohl als Sinnbilder, als auch traditionell als achtstelliger Dateiname mit Suffix dargestellt. In der Kopfleiste werden Datum und Uhrzeit eingeblendet. Ansonsten aber hat sich die Erscheinungsweise gegenüber Windows nicht geändert. PC-DOS 4.0, MS-DOS 4.0, Microsoft Windows und der OS/2 Presentation Manager entsprechen alle den SAA-Vorgaben der IBM.

Christian Wedell, Geschäftsführer der Microsoft GmbH, hierzu: »Mit der neuen Version des Betriebssystems MS-DOS 4.0 bzw. PC-DOS 4.0 haben wir nun vom einfachen PC über Windows-Rechner bis hin zu neuen OS/2-Rechnern eine durchgehende SAA-Oberfläche mit weitgehend identischer Bedienbarkeit. Gemeinsam mit IBM werden wir diese Oberflächenstandards auf SAA-Basis ständig weiterentwickeln. So wird eines Tages sicherlich auch Microsoft Windows über die File-Strukturdarstellung des Presentation Managers verfügen. Diese Planungen und Weiterentwicklungen setzen wir völlig unberührt von den juristischen Klärungen über Copyrights mit der Firma Apple fort.«

Reiner Doelle, Leiter Produktmanagement PC der IBM Deutschland: »DOS 4.0 und der OS/2 Presentation Manager wurden von IBM und Microsoft gemeinsam im Hinblick auf eine Integration in unser SAA-Konzept entwickelt. SAA ist für beide Firmen der Oberflächenstandard der nächsten Jahre.«

Christian Wedell: »Im gemeinsamen Standardisierungsinteresse von IBM und Microsoft liegt es, wenn Microsoft sein Produkt Windows oder vielleicht eines Tages auch seine Works-Oberfläche der gemeinsamen Entwicklung anpaßt. Weitgehend identisch ist das Erscheinungsbild dieser Produkte bereits heute.«

Die Brücke zwischen IBM und Apple

Mit der Ankündigung einer PC-Software-Version von Microsoft Mail für den Herbst bahnt sich eine bedeutende Verbindung zwischen IBM-PC-Rechnern und den Apple-Rechnern auf der Basis des AppleTalk-Netzwerks an. Der Empfang und Versand von Mitteilungen, Grafiken und ganzen Disketten-Inhalten ist damit sowohl von PC zu PC als auch zu und von Macintosh-Rechnern möglich. Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH: »Die neue Microsoft-Software wird die immer wiederkehrende Diskussion 'Apple oder IBM' sicher abflauen lassen, da nun die Verbindung zwischen beiden Systemen möglich wird. Auch zwischen beiden Dateiformaten wird Microsoft Mail für Verständigung sorgen.«

Die für IBM PCs und Kompatible einfach installierbare Software berücksichtigt die bekanntesten Steckkarten für den PC am AppleTalk-Netzwerk, wie die Karten von Apple, TOPS und Tangent.

Da Microsoft Mail die gleiche Windows-ähnliche Oberfläche wie Microsoft Works hat, bleibt dem PC-Anwender die gewohnte Arbeitsumgebung auf dem Bildschirm erhalten. Im einzelnen enthält Microsoft Mail Features wie Server-Funktion im Hintergrund-Betrieb, Mehrbereichs-Unterstützung, Datei-Übertragung, Netzwerk-Leistungen, ein stets verfügbares On-Line-Tutorial sowie volle Mailbox-Sicherheit durch Paßwort-Schutz.

Microsoft Mail für PC im AppleTalk-Netz unterstützt voll die Arbeitsumgebung von AppleShare und MultiFinder. Microsoft beabsichtigt die Anbindung von Microsoft Mail an Microsoft Word 4.0, damit der Word-Anwender Mitteilungen empfangen, verarbeiten und senden kann, ohne sein Textprogramm zu verlassen. Microsoft Mail für PC ist in der englischen Version ab 1. Quartal 1989 verfügbar.

Nun gibt es QuickBasic auch für den Apple Macintosh

Mit Microsoft QuickBasic steht nun auch Apple Macintosh-Anwendern die vielseitige Leistungsfähigkeit dieses Entwicklungs-Werkzeugs zur Verfügung, in dem die Vorteile eines Basic-Interpreters mit denen eines Basic-Compilers kombiniert wurden.

Die Interpreter-Vorteile liegen in der Programm-Erstellung mit Editor und Debugger. Die Vorteile des Compilers machen sich hingegen in der Schnelligkeit der Programm-Ausführung bemerkbar. In QuickBasic geschriebene und compilierte Programme sind eigenständige Applikationen, die nach ihrer Fertigstellung weder Interpreter noch Compiler zum Ablauf benötigen.

Microsoft QuickBasic eröffnet für den Macintosh neue Dimensionen. Dabei ist die einfach beherrschbare Sprache Basic sowohl für den professionellen Anwender als auch den Amateur eine interessante Alternative in der Programmierung. Selbst komplexe Aufgaben werden leicht lösbar unter der neuen, verbesserten Bedienungsoberfläche, die der vom Macintosh gewohnten Umgangsweise entgegenkommt. Zu den besonderen QuickBasic-Eigenschaften gehört die Unterstützung für musikalische und farbgrafische Aufgaben. Hinzu kommt, daß QuickBasic die Fähigkeiten der Prozessoren 68020 und 68881 vollkommen nutzt.

Mitteilungen .... Mitteilungen .... Mitteilungen

Durch die Kombination von Compiler und Interpreter auf einer gemeinsamen Bedienungsoberfläche lassen sich Kompilierung und Ausführung eines Programms auf einen einfachen Klick mit der Maus reduzieren. Der Zugriff auf die Macintosh-Toolbox, eine Subroutinen-Bibliothek, gewährleistet ferner, daß Anwendungsprogramme auf einfache Weise mit Macintosh-Standard-Interfaces versehen werden und gewohntes Aussehen sowie bekannte Bedienungs-Vorteile erhalten. Hierzu ist lediglich die Übernahme gewünschter Prozeduren aus der Bibliothek notwendig.

Die Eigenschaften des enthaltenen Quellcode-Debuggers sind geprägt durch Einzelschritt-Abarbeitung, Trace-Funktion und die Möglichkeit, Unterbrechungen zu setzen. Im Direkt-Modus können nach jeder Programmunterbrechung die Inhalte von Variablen angezeigt sowie Kommandos unabhängig vom Programm ausgeführt werden. Die jederzeit verfügbare »On-Line-Hilfe« von QuickBasic informiert auf Knopfdruck über Basic-Vokabeln, Bibliotheks-Funktionen und Aufrufe, die zu ROM-Routinen gehören.

Zum Lieferumfang gehören zahlreiche Programm-Beispiele, die den Einstieg in Basic erleichtern und viele der QuickBasic-Besonderheiten anschaulich nahebringen. QuickBasic erfordert in dieser modernsten Version 1 MegaByte RAM im Macintosh Plus, SE oder II und mindestens ein doppelseitiges 800 KByte-Laufwerk. Die Verwendung einer Festplatte oder eines zweiten 800-Kbyte-Laufwerks ist empfehlenswert. Microsoft QuickBasic für den Macintosh ist ab Oktober 1988 verfügbar.

Microsoft liefert MS-DOS/CD-ROM-Extensions 2.0 aus

Unlängst gab Microsoft die Auslieferung der Version 2.0 der MS-DOS/CD-ROM-Extensions (MSCDEX, Version 2.0) bekannt. Die neue Version ist in der Lage, CD-ROM-Disks zu lesen, die sowohl im Original-»High Sierra«-Format als auch in der neueren Version dieses Formates, dem ISO9660-Format, gepreßt sind. Damit können alle Computer unter MS-DOS mit MSCDEX 2.0 auf Files zugreifen, die auf beliebigen CD-ROM-Disks gespeichert sind. Applikationsentwickler sind nun nicht mehr gezwungen, spezielle Software für unterschiedliche CD-ROM-Laufwerke zu entwickeln. »Noch wichtiger ist es vielleicht«, so Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, »daß die Anwender sich nun keine Gedanken mehr um Datenträger-Austauschbarkeit bei unterschiedlichen CD-ROM-Disketten und Laufwerken mehr machen müssen«.

MS-DOS/CD-ROM-Extensions 2.0 unterstützt den japanischen Kanji-Zeichensatz, und zwar sowohl was die File-Namen als auch was die Files selber betrifft. Es ist nun möglich, Audio-Daten, Texte und Grafiken zu integrieren. Bei zukünftigen Applikationen ist der Vorteil der Multimedia-Fähigkeit von CD-ROMs, die darin besteht, Ton, Grafik und Text auf einem einzigen Medium zusammenzuführen, voll nutzbar. MS-DOS/CD-ROM-Extensions 2.0 läuft auf PC mit MS-DOS 3.1 oder einer höheren Version.

Mehr als eine Million »Mäuse« verkauft

In diesen Tagen überschritt Microsoft die Millionen-Grenze beim Verkauf der berühmten Microsoft-Maus, die ja vielfach als das »Original« bezeichnet wird. Seit Juli 1986 haben sich die Verkaufszahlen dieses Produktes nahezu dramatisch entwickelt: In knapp 12 Monaten sprang der Absatz von 500.000 Stück auf nunmehr über eine Million. Aus diesem Grunde erwartet Microsoft, daß schon gegen Ende dieses Sommers die 1 1/2-Millionen-Marke erreicht wird.

Vor wenigen Wochen stellte Microsoft eine völlig neuentwickelte Maus vor, die neben einem exzellenten Design auch für Komfort und leichte Handhabung in Kombination mit einer hochqualitativen Grafikunterstützung und Software steht.

Die Marktdaten zeigen, daß sich der Microsoft-Maus-Absatz durch die neue Maus - die übrigens einen Preis für die besonders gelungene Formgebung erhielt - noch beschleunigen wird und auch der Einsatz von Grafik-Software sowie die Verfügbarkeit von Software-Produkten, die Mausunterstützung bieten, erheblich steigen.

»Microsoft wird auch weiterhin der Marktführer in diesem Markt bleiben, weil wir verstehen, was die Anwender benötigen, um den vollen Nutzen aus grafik- und zeichenorientierten Programmen zu ziehen«, so Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH.

Mit dem wachsenden Einsatz von Grafikprogrammen und der Vorstellung leistungsfähiger grafischer Bedienungsoberflächen wie Microsoft Windows und MS-OS/2 Presentation Manager, hat auch der Bedarf für Microsoft-Mäuse entsprechend zugenommen. Auch verbesserte Hardware wie die neuen VGA-Monitore, haben dabei eine Rolle gespielt, weil sie dem Anwender die Möglichkeit bieten, die neuen grafischen Funktionen voll zu nutzen. Der Trend, Mausunterstützung sowohl für grafik- wie auch für zeichenorientierte Software zu bieten, nimmt ebenfalls zu. Software-Entwickler erkennen in zunehmendem Maße, in welcher Weise eine Maus die Anwender-Produktivität verbessern kann. Deshalb unterstützen bislang mehr als 250 Software-Pakete die Microsoft-Maus.

Die neue Microsoft-Maus ist in einer Reihe unterschiedlicher Software-Pakete im Handel erhältlich. Eine dieser Versionen ist z.B. für das Programm »Paintbrush« ausgelegt, ein Zeichenprogramm, das die Firma Z-Soft für die MS-DOS-Betriebssystem-Umgebung entwickelt hat. Der empfohlene Verkaufspreis dieser Version wird 390 Mark zuzüglich Mehrwertsteuer betragen. Eine andere Version, die zu einem empfohlenen Preis von 650 Mark zu-

Mitteilungen .... Mitteilungen .... Mitteilungen

züglich Mehrwertsteuer angeboten wird, umfaßt Microsoft Windows 2.0 und Z-Soft PC-Paintbrush für Windows. Die dritte Version, mit einem empfohlenen Verkaufspreis von 525 Mark zuzüglich Mehrwertsteuer, wird zusammen mit EasyCad, einem Programm der Firma Evolution Computing, angeboten. EasyCad ist ein leistungsfähiges CAD-Programm. Alle Versionen enthalten die Software »Mouse Menus«.

Die HP-Newwave-Applikationsumgebung wird von Microsoft im Rahmen zukünftiger Excel-Versionen unterstützt

Microsoft und Hewlett-Packard haben in diesen Tagen eine Vereinbarung getroffen, auf deren Basis die beiden Unternehmen gemeinsam das Tabellen-Kalkulationsprogramm Microsoft Excel für die HP-NewWave-Umgebung entwickeln und vermarkten wollen.

HP-NewWave ist eine fortschrittliche Software-Applikationsumgebung, die auf der grafischen Bedienungsoberfläche Microsoft Windows, Version 2.0, basiert. Sie erlaubt dem Anwender das Arbeiten mit mehreren Applikationsprogrammen und vereinfacht das Manipulieren von Daten aus verschiedenen Quellen.

HP-NewWave verbessert darüber hinaus die PC-Integration, indem das Programm dem Anwender einen sofortigen Überblick über die gesamten Informationsressourcen in einem Netzwerk gibt.

»Mit den leistungsfähigen, systemweiten Diensten, die HP-NewWave bietet, macht Microsoft Excel einen weiteren Schritt vorwärts zum leistungsfähigsten, flexibelsten und am einfachsten zu handhabenden Tabellen-Kalkulationsprogramm, das es auf dem Markt gibt«, so die Stellungnahme von Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, zu dem Abkommen zwischen Microsoft und Hewlett-Packard.

Robert J. Frankenberg, General Manager der Information Systems Group von Hewlett-Packard, meint, daß aus Sicht seines Unternehmens die Vereinbarung mit Microsoft einen großen Fortschritt in der Akzeptanz von HP-NewWave in Software-Entwickler-Kreisen bewirkt: »Wir sind sehr erfreut, ein Produkt der neuen Software-Generation wie Microsoft Excel zu haben, das an die HP-NewWave-Umgebung angepaßt ist und glauben, daß sich viele HP-NewWave-Kunden für dieses Tabellen-Kalkulationsprogramm entscheiden werden.«

Die Verbesserungen bezüglich HP-NewWave werden nach einer Darstellung von Microsoft auch in das Microsoft Excel Standard-Produkt integriert, so daß Anwender keine spezielle Version erwerben müssen, wenn sie mit HP-NewWave arbeiten. Die neue Version von Microsoft Excel wird den PC-Anwendern Zugriff auf die wichtigsten Funktionen von HP-NewWave, einschließlich der HP-Objekt-Management-Funktion und »Agents«, einer Funktion zur Automatisierung systemweiter Tasks, verschaffen. Die neue Microsoft Excel-Version wird darüber hinaus eine Plattform zur Implementierung von KI-Funktionen (Künstliche Intelligenz) sein.

Die Objekt-Management-Funktion nutzt die Möglichkeit des dynamischen Datentransfers bei Microsoft Excel, so daß der Anwender »Hotlinks« oder Brücken zwischen Microsoft Excel und anderen Applikationsprogrammen einrichten kann, die unter HP-NewWave laufen. Einmal eingerichtete Verbindungen werden kontinuierlich durch HP-NewWave verwaltet und gestatten die gemeinsame Nutzung von Daten, die auch automatisch in den verschiedenen Programmen und Dateien aktualisiert werden. Durch Einsatz der Objekt-Management-Funktion werden Daten aus Microsoft Excel-Tabellen zu Objekten, die auf einfache Weise mit Objekten aus anderen Programmen zusammengeführt werden können, um daraus ein Dokument zu erstellen.

Das im November 1987 vorgestellte HP-NewWave-Entwicklungskit wird nun an alle Interessenten ausgeliefert, die Applikationen für die HP-NewWave-Umgebung schreiben wollen. Die Anwendungsumgebung wird im späteren Verlauf dieses Jahres verfügbar sein. Eine Unterstützung für HP-NewWave/Microsoft Excel ist ab Mitte 1989 lieferbar. Die Firma Hewlett-Packard plant, eine Vertriebslizenz für Microsoft Excel zu erwerben, so daß der Verkauf auch über den Vertriebsapparat von HP erfolgt.

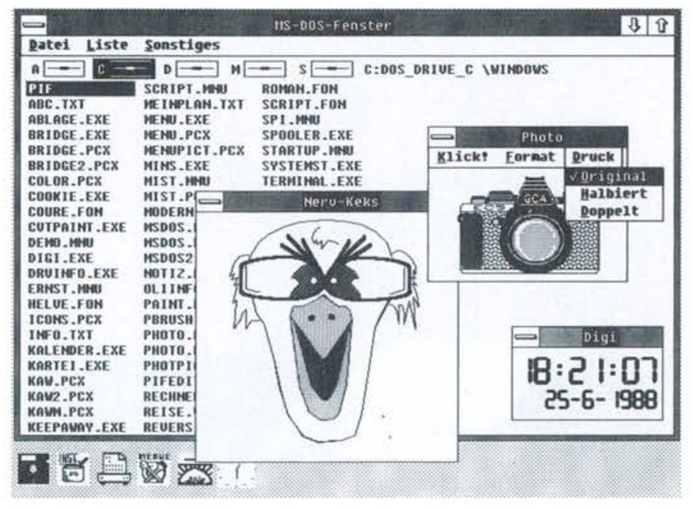
Hewlett-Packard und Microsoft haben die Absicht, jeweils vergleichbare Funktionen und Möglichkeiten für die zukünftigen Versionen von Microsoft Excel und HP-NewWave zu entwickeln, die auf dem Microsoft-Multitasking-Betriebssystem OS/2 und der grafischen Bedienungsoberfläche Presentation Manager laufen.

Accessories No.1: Toolbox für Microsoft Windows

Die Gesellschaft für Computeranwendungen GCA, stellt mit Accessories No.1 jetzt eine »Toolbox« für den Anwender von Microsoft Windows zur Verfügung. Accessories No.1 ist ein Produkt langjähriger Erfahrungen des Ludwigsburger Unternehmens aus der Programmierung mit Microsoft Windows/Presentation Manager, dem Applikations-Trägersystem der Zukunft.

Andere für Microsoft Windows verfügbare Applikationen, wie z.B. Pagemaker oder Excel, haben sich erfolgreich am Markt etabliert. Die in der Praxis dringend benötigten kleinen Anwendungen, Utilities und Hilfsprogramme sind jedoch unberücksichtigt geblieben. Diese Lücke schließt jetzt Accessories No.1. Es ist die erste Kollektion aus einer Serie von Programmen für Microsoft Windows. Ziel dieser Programme ist die sinnvolle Integration von Daten und Anwendungen unter Microsoft Windows.

Mitteilungen .... Mitteilungen .... Mitteilungen



Sieben Anwendungen in der neuen Toolbox

Das neue System bietet dem Endanwender Programme zur Gestaltung individueller, grafischer Menüsysteme, zur Konvertierung von GEM-Dateien in Window-Formate, zur Abspeicherung und Bearbeitung verschiedenster Bildschirminhalte, Sicherung des eigenen Systems gegen unberechtigten Zugriff sowie Analyse des Druckers und der Druckertreiber. Den Leistungsumfang runden eine Digitaluhr und ein kleineres »Unterhaltungsprogramm« ab.

Das mit dem Paket gelieferte Menüsystem MENU optimiert das Konzept einer klaren, durchgängigen, einfach zu bedienenden Benutzeroberfläche. Das Menüsystem erlaubt die Gestaltung individueller graphischer Menüs. Graphische Symbole (Icons) stehen stellvertretend für Programme oder Makros. Die Aktivierung der einzelnen Menüoptionen erfolgt durch Anklicken mit der Maus oder durch Selektion über die Tastatur.

BRIDGE macht die Konvertierung von GEM nach MS-Windows möglich. Alle GEM-Dateiformate werden automatisch in äquivalente MS-Windows-Formate umgesetzt. Konvertierte Dateien bleiben editierbar und können dann unter MS-Windows weiterbe- und verarbeitet werden. Dies gilt selbstverständlich für alle Arten von Bild-, Pixel- und Textdateien.

PHOTO ist ein Werkzeug für die Dokumentation oder zur Bildübernahme von Programmen ohne direkte Datenaustauschmöglichkeit.

Bildschirmausschnitte oder ganze Screen-Inhalte können mit PHOTO einfach abgespeichert und weiterbearbeitet, ausgedruckt oder in andere Programme übernommen werden. PHOTO ist kompatibel zur gesamten von MS-Windows unterstützten Hard- und Software.

KEEPAWAY bietet dem Anwender einen Passwortschutz für das laufende System, wobei im Hintergrund alle arbeitenden Programme ungestört weiter ablaufen.

Mit dem Auskunfts- und Analyseprogramm DRVINFO, kann der Anwender die immer wieder auftretenden Unklarheiten bezüglich Druckern und Druckertreibern erkennen und beseitigen, da das Programm alle Informationen über den Drucker und die vom Treiber unterstützten Funktionen abrufbar hält.

Als platzsparende Alternative zu UHR.EXE aus MS-Windows ist in Accessories No.1 die Digitaluhr DIGI mit Datum enthalten.

Das siebte Programm in der Toolbox ist COOKIE. Nervenaufreibend und unsinnig bringt es einfach Spaß bei der Arbeit am Rechner.

Die ausführliche Dokumentation des Produkts und die im Lieferumfang enthaltenen Installationsprogramme erlauben es jedem Anwender, die weitgehend selbsterklärenden Module der neuen Toolbox sofort unter MS-Windows einzusetzen.

Mitteilungen

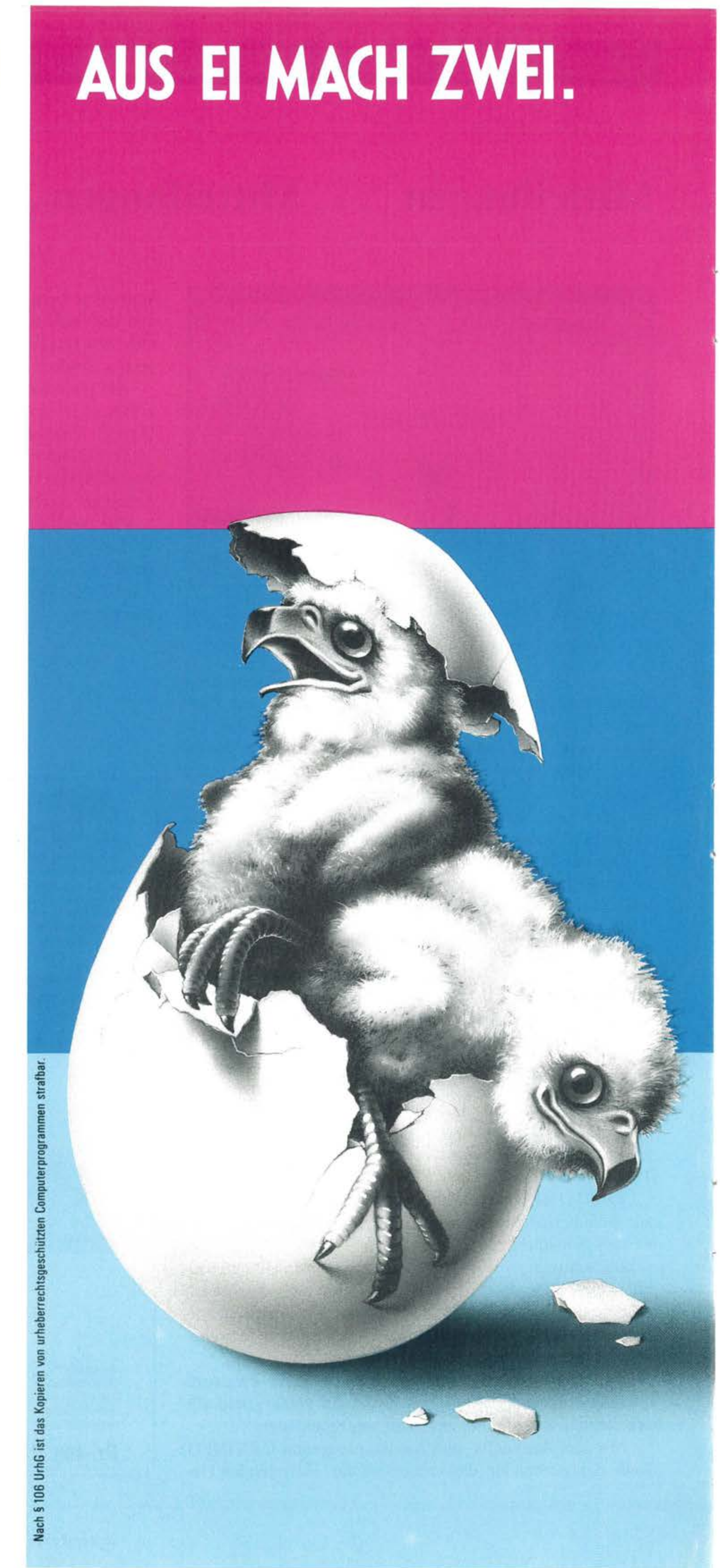
Microsoft Works Version 2.0 für den Macintosh

Mit der für den Herbst vorgesehenen Auslieferung der englischen Version von Microsoft Works 2.0 werden wichtige Programmverbesserungen für den Macintosh verfügbar. Microsoft Works Version 2.0 vereint fünf unterschiedliche Aufgaben: Textverarbeitung, Tabellenkalkulation, Datenbank, Kommunikation und Grafik in einem einzigen Programmpaket.

Microsoft Works, das sich in kurzer Zeit seinen Platz unter den drei Spitzenprodukten dieser Kategorie für den Macintosh erobern konnte, bietet in der Version 2.0 eine Reihe zusätzlicher Leistungen. So wurden die grafischen Fähigkeiten erweitert und verbessert. Ferner sorgen Makros für die Automatisierung stets wiederholter Aufgaben. Ein integriertes Rechtschreibprogramm ergänzt die Textverarbeitung. Außerdem können umfangreichere Spreadsheets mit allen Informationen einschließlich einer Datum-/Zeitfunktion verarbeitet werden.

Die leichte Erlernbarkeit, die Works bekanntlich auszeichnet, wird durch ein CBT(Computer Based Training)-Lernprogramm, das am Bildschirm angewählt werden kann, zusätzlich unterstützt.

Microsoft Works Vielseitigkeit, die einen schnellen Einstieg in die professionelle Anwendung ermöglicht, enthält für den Bereich der Grafik beziehungsweise Geschäftsgrafik zahlreiche Hilfsmittel zum Zeichnen von Linien, Kreisen, Rechtecken, Winkeln und Polygonen sowie zum freihändigen Zeichnen. Diese grafischen Fähigkeiten können sowohl im Spreadsheet als auch in der Textverarbeitung angewendet werden. Die einzelnen Gruppen und Elemente einer Zeichnung können zur nachträglichen Bearbeitung selektiert werden.



Mitteilungen

Drei weitere Leistungen stehen allen fünf Programmen innerhalb des Pakets zur Verfügung: Farbunterstützung für den Macintosh II, Druckvorschau am Bildschirm zur Begutachtung einer Seite vor dem Ausdruck sowie ein Makrorecorder, der zur Aufzeichnung eines Arbeitsablaufs zum Zweck der späteren wiederholten Ausführung dient.

Eine in der Textverarbeitung enthaltene Rechtschreibprüfung kann auf ein Wörterbuch zurückgreifen, das auf die eigenen Bedürfnisse abgestimmt und erweitert werden kann. Verbesserungen der Mailmerge-Funktionen in der Textverarbeitung erlauben die Verwendung mehrfacher Aufzeichnungen, schließen die Zusammenfassung von Textdateien und Datenbankauszügen ein und vereinfachen die Verwendung mehrspaltiger Adreßaufkleber.

Das Tabellenkalkulationsmodul erlaubt bei Works 2.0 nun 256 Spalten und bis zu 16.382 Zeilen. Datum-/Zeitfunktionen dürfen enthalten sein. Neben den Daten der einzelnen Zellen kann man nun auch Informationen über jede einzelne Zelle speichern. Das Datenbankmodul wurde durch Einfügen von Datum-/Zeitfunktionen und durch flexiblere Auswertungsfähigkeiten verbessert.

Datenimport und -export verschaffen Microsoft Works auf dem Macintosh Kompatibilität zu Microsoft Word und Microsoft Excel durch Dateien im RTF- sowie SYLK-Format.

Microsoft Works 2.0 für den Macintosh wird in englischer Version ab Oktober 1988 verfügbar sein. Eine deutsche Version wird voraussichtlich im 1. Quartal 1989 in den Handel kommen.

Eine einfache Utility erleichtert die Windows-Programmierung:

Windows-Programme effektiver debuggen

Das Debuggen ist ein unverzichtbarer und unvermeidbarer Schritt bei der Entwicklung jedes Programms. Bei normalen MS-DOS-Programmen beginnt das Debuggen gewöhnlich mit dem Einbau zahlloser Print-Anweisungen in das Programm. Wenn das nicht hilft, kommt es zur Eskalation der Schlacht und zu Sitzungen mit DEBUG, CodeView oder sogar einem Hardware-Debugger.

Unglücklicherweise funktionieren viele der traditionellen Test-Techniken in der Windows-Umgebung nicht. In den letzten Monaten hat es gewaltige Verbesserungen in Bezug auf das Debuggen unter Windows gegeben, besonders durch die Ankündigung von CodeView für Windows (siehe separaten Artikel). Doch die Programmierer kämpfen immer noch mit der schwierigen Aufgabe des Debuggens und Feilens an ihren Windows-Anwendungen.

Methoden des Debuggens

Theoretisch gibt es viele Arten, Debug-Methoden zu klassifizieren. Häufig wird das Debuggen in drei Kategorien eingeteilt, die invasiven, die nichtinvasiven und die kombinierten Techniken.

Beim invasiven Debuggen wird normalerweise der Quellcode der Anwendung geändert und neu kompiliert. Die Debug-Tools und ihre Einsprungspunkte werden ganz einfach Teil des Programms, wodurch diese Technik sehr leicht zu implementieren ist. Zwei Beispiele für diese Technik sind fest codierte Meldungsboxen und Ausgabeanweisungen, die auf den Bildschirm oder eine temporäre Datei umgeleitet werden.

Diese Methode reicht oft aus, um einen großen Anteil der Codierungs- und Algorithmen-Fehler abzufangen. Bei geschickter Anwendung kann diese Methode das Haupt-Debugging-Tool für große und komplizierte Anwendungen sein.

Diese Methode macht es jedoch notwendig, daß man Zugriff auf den fehlerhaften Quellcode hat, was ein Problem ist, wenn man eine Anwendung testet, die auf ein Programm reagiert, das man nicht selbst geschrieben hat. Zusätzlich wird durch den Einbau der Debug-Anweisungen oder Tools in die Anwendung der Charakter des Problems verändert, wodurch die Situation weiter verkompliziert wird. Dies erweist sich besonders dann als nachteilig, wenn man es mit systemnahen Dingen wie der Speicherverwaltung und der Kommunikation zwischen Programmen zu tun hat.

Beim nichtinvasiven Debuggen ist keine Änderung oder Neukompilierung einer Anwendung notwendig. Das Debug-Tool und seine Steuerfunktionen stehen außerhalb der Anwendung und können gewöhnlich bei einem existierenden Programm verwendet werden. Beispiele für diese Technik sind das Aufzeichnen von Meldungen, Eingriffe für

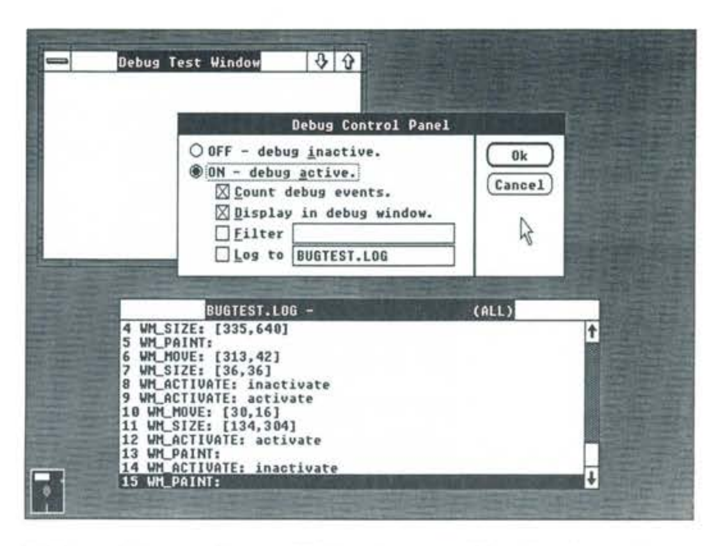


Bild 1: *Mit der Debug-Utility können Sie die Operationen einer Anwendung protokollieren und untersuchen.*

die Protokollierung, Alias-Bibliotheken, Profiler und sogar Hardware-Debugger.

Wenn die nichtinvasive Methode korrekt eingesetzt wird, kann sie eine Menge extern erhaltener Diagnoseinformationen zur Verfügung stellen. Diese Methode ist besonders beim Debuggen komplexer Verbindungen zwischen Anwendungen nützlich. Des weiteren wird mit externen Debug-Tools die eventuelle Aufsummierung von Fehlern eingeschränkt.

Auf der negativen Seite ist zu sagen, daß diese Methode oft kompliziert ist: Es wird nur über extern wahrnehmbare Ereignisse berichtet und sie tendiert dazu, viele unwichtige Informationen zu erzeugen, die die wenigen wichtigen Daten umgeben. Der Programmierer muß gewöhnlich die mitgeteilten Ereignisse interpretieren und nach dem suchen, was wirklich wichtig ist.

Beim kombinierten Debuggen werden sowohl die invasive als auch die nichtinvasive Methode verwendet. In den meisten Fällen braucht das Programmm nur mit einem anderen Compilerschalter neu kompiliert werden, der Symboldateien oder andere für das Debug-Tool nützliche Informationen erzeugt. Beispiele für diese Methode sind CodeView, SymDeb und andere, kompliziertere Hardware-Debugger, die die Interna von Anwendungen handhaben können.

Diese Methode stellt wahrscheinlich die leistungsfähigste dieser Debug-Techniken dar, denn damit können sogar heimtückische Programmierweisen und Systemfehler festgestellt und beschrieben werden. Zusätzlich kann der Programmierer beim Testen gewöhnlich die internen Datenstrukturen untersuchen, die einem einen noch besseren Überblick über eine bestimmte Situation ermöglichen.

Um diese Methode effektiv einzusetzen, muß der Programmierer intime Kenntnisse des verwendeten Betriebssystems und der Hardware haben. Die Leistungsfähigkeit dieser Methode wird dadurch geschmälert, daß sie sehr

```
/*
 * WINDOWS 2.00 DEBUG UTILITY - HEADER FILE
 *
 * LANGUAGE : Microsoft C 5.0
 * MODEL    : small or medium
 * STATUS   : operational
 *
 * 12/11/87 1.00 - Kevin P. Welch - initial creation.
 *
 */

/* debug control panel constants */
#define BUG_OFF          100 /* debug inactive */
#define BUG_ON           101 /* debug active */
#define BUG_COUNT        102 /* count debug events */
#define BUG_DISPLAY      103 /* display debug events */
#define BUG_FILTER       104 /* filter debug events */
#define BUG_FILTERLIST   105 /* debug filter list */
#define BUG_LOG          106 /* log debug events */
#define BUG_LOGFILE      107 /* debug log file */
#define BUG_OK           108 /* OK button */
#define BUG_CANCEL       109 /* CANCEL button */

/* default dialog box definitions */
#define DEF_BUTTON       (BS_DEFPUSHBUTTON|WS_TABSTOP|WS_CHILD)

/* standard dialog box definitions */
#define STD_FRAME        (SS_BLACKFRAME|WS_CHILD)
#define STD_CHECKBOX     (BS_CHECKBOX|WS_TABSTOP|WS_CHILD)
#define STD_BUTTON       (BS_PUSHBUTTON|WS_TABSTOP|WS_CHILD)
#define STD_RADIO        (BS_RADIOBUTTON|WS_TABSTOP|WS_CHILD)
#define STD_EDITFIELD
(WS_BORDER|WS_TABSTOP|WS_CHILD|ES_AUTOHSCROLL)

/* debug function definitions */
BOOL FAR         Debug();
BOOL FAR PASCAL DebugControl( HWND );
BOOL FAR PASCAL DebugSetup( HWND, WORD, WORD );
BOOL FAR PASCAL DebugControlDlgFn( HWND, WORD, WORD, LONG );
```

Listing 1: *Die Header-Datei der Debug-Utility.*

lernintensiv ist und sehr viele Informationen erzeugt. Viele Programmierer zögern, sie zu verwenden, außer bei den unübersichtlichsten und schwierigsten Problemen.

Eine Windows-Debug-Utility

Nach der Entwicklung zahlreicher Windows-Anwendungen und dem Kampf mit den Problemen des Debuggens entschied ich mich, ein einfaches invasives Debug-Tool zu entwickeln, das mir einen sinnvollen Überblick über die Operationen ermöglichen sollte, die in einer fehlerhaften Anwendung durchgeführt werden.

Meine Bemühungen führten zur Windows-Utility Debug, einem kleinen Objekt-Modul (kompiliert ca. 4 Kbyte), das auf Wunsch zu einem Programm hinzugelinkt werden kann. Es bietet in der Windows-Umgebung ähnliche Debug-Möglichkeiten wie mit Ausgabeanweisungen.

Sobald Debug in Ihre Anwendung integriert ist, können Sie die Werkzeuge der Utility dazu verwenden, zahlreiche interne Programmereignisse anzuzeigen oder aufzuzeichnen. Der ganze Debug-Prozeß wird über eine zentrale Steuertafel kontrolliert, die über das Systemmenü der Anwendung zu erreichen ist (*Bild 1*). In dieser Steuertafel kann man:

```
/*
 * WINDOWS 2.00 DEBUG UTILITY - SOURCE CODE
 *
 * LANGUAGE : microsoft C 5.0
 * MODEL    : small or medium
 * STATUS   : operational
 *
 * 12/11/87 1.00 - Kevin P. Welch - initial creation.
 *
 */

#include <windows.h>
#include <string.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include "debug.h"

/* internal macros */
#define TOGGLE(x,y)CheckDlgButton(x,y,!IsDlgButtonChecked(x,y))
#define ENABLE(x,y,z)   EnableWindow(GetDlgItem(x,y),z)

/* general definitions */
#define SHARED          S_IREAD|S_IWRITE
#define MODE_APPEND O_RDWR|O_APPEND|O_BINARY
#define MODE_CREATE O_RDWR|O_CREAT|O_BINARY

/* listbox viewport definitions */

#define PORT_X          20  /* viewport x origin */
#define PORT_Y          40  /* viewport y origin */
#define PORT_WIDTH  460 /* viewport width */
#define PORT_HEIGHT 140 /* viewport height */

/* debug statement data structure */

#define MAX_FILTER  32  /* maximum filter categories */

typedef struct {

    /* general options */
    BOOL     bSetup;        /* debug setup flag */
    BOOL     bActive;       /* debug active flag */

    /* count options */
    BOOL     bCount;        /* count events flag */
    LONG     lMsgCount;     /* debug message count */

    /* display options */
    BOOL     bDisplay;          /* display debug events */
    HWND     hListBox;          /* listbox window handle */
    WORD     wEntries;          /* current listbox lines */
    WORD     wMaxEntries;       /* maximum listbox lines */

    /* filter options */
    BOOL     bFilter;           /* filter debug events */
    WORD     wFilterSize;       /* size of categories */
    WORD     Filter[MAX_FILTER]; /* list of categories */

    /* log-file options */
    BOOL     bLog;              /* log debug events */
    char     szLogFile[64];     /* debug log file name */

} OPTIONS;

/* global data definitions */
static OPTIONS  Options;


/*
 * DebugSetup( hWnd, wMsg, wMax ) : bResult
 *
 *      hWnd     main application window handle
 *      wMsg     debug control panel system message
 *      wMax     maximum number of debug lines
 *
 * This function is responsible for creating the debug viewport
 * window class and for attaching the control panel to the
 * system menu of the window handle provided (this implies that
 * the window handle provided has a system menu).
```

```
 *
 * A boolean value of TRUE is returned if the setup process was
 * successful.
 *
 */

BOOL FAR PASCAL DebugSetup( hWnd, wMsg, wMax )
    HWND    hWnd;
    WORD    wMsg;
    WORD    wMax;
{
    /* local variables */
    BOOL        bOk;            /* boolean result */
    HMENU       hSysMenu;       /* temporary menu handle */
    WNDCLASS    WndClass;       /* temporary class structure */
    HANDLE      hInstance;      /* handle to window instance */
    char        szModule[64];/* main module file name */

    /* boolean result */
    bOk = FALSE;
    hInstance = GetWindowWord( hWnd, GWW_HINSTANCE );

    /* check if debug already activated */
    if ( !Options.bSetup ) {

        /* retrieve system menu handle */
        hSysMenu = GetSystemMenu( hWnd, FALSE );
        if ( hSysMenu ) {

            /* change system menu to reflect control panel */
            ChangeMenu( hSysMenu, 0, NULL, 999,
                MF_APPEND|MF_SEPARATOR );
            ChangeMenu( hSysMenu, 0, "&Debug...", wMsg,
                MF_APPEND|MF_STRING);

            /* create listbox window -initially hidden */
            Options.hListBox = CreateWindow(
                "LISTBOX",          /* class name */
                "",                 /* window caption */
                WS_BORDER|WS_VSCROLL,
                                    /* window style */
                PORT_X,             /* x position */
                PORT_Y,             /* y position */
                PORT_WIDTH,         /* width */
                PORT_HEIGHT,        /* height */
                NULL,               /* parent window */
                NULL,               /* menu */
                hInstance,          /* instance handle */
                (LPSTR)NULL         /* other data */
                );

            /* continue if successful */
            if ( Options.hListBox ) {

                /* define options data */
                bOk = TRUE;
                Options.bSetup = TRUE;
                Options.wMaxEntries = wMax;

                /* define default log file name */
                GetModuleFileName( hInstance,
                szModule, sizeof(szModule) );

                *strrchr(szModule,'.') = 0;
                sprintf(
                    Options.szLogFile,
                    "%s.LOG",
                    (char *)(strrchr(szModule,'\\')+1)
                );

            }

        }

    }

    /* return final result */
    return( bOk );

}
```

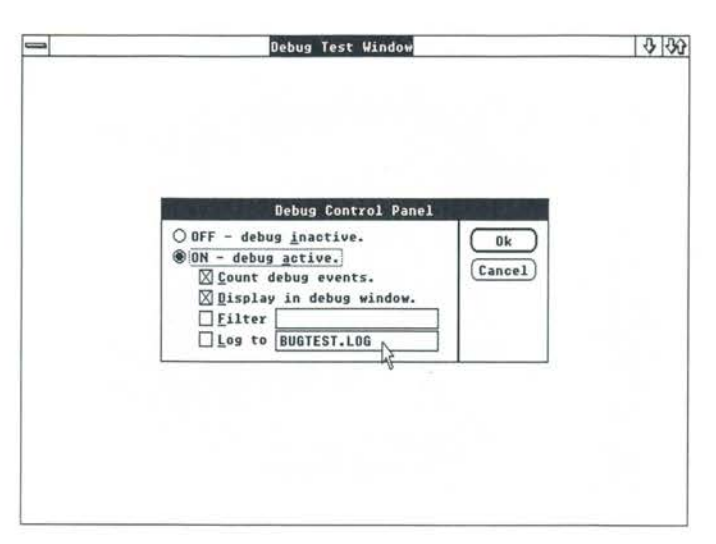


Bild 2: *Die Steuertafel der Debug-Utility.*

- das Debuggen ein- oder ausschalten,
- unerwünschte Debug-Ereignisse herausfiltern,
- die Debug-Ausgabe auf dem Bildschirm ausgeben oder in einer Datei speichern.

Funktionen der Debug-Utility

Das Debug-Modul, das zu Ihrer Anwendung hinzugelinkt wird, besteht aus zwei Dateien, `DEBUG.H` und `DEBUG.C` (*Listings 1 und 2*). Das Debug-Modul besteht aus vier Funktionen, von denen drei in Ihrer Anwendung explizit verwendet werden können.

Die erste Funktion ist `DebugSetup`, die für die Anlage des Debug-Fensters und das Hinzufügen des Debug-Befehls zum Systemmenü der Anwendung zuständig ist. Diese Funktion muß mit der Fenster-Handle, einer Meldung und der maximalen Anzahl Zeilen, die von Debug verwaltet werden sollen, aufgerufen werden.

Die Funktion `DebugSetup` nimmt an, daß es im Fenster der übergebenen Handle ein Systemmenü gibt. Wenn das Hauptfenster kein Systemmenü enthält und sie auf andere Weise auf die Debug-Steuertafel zugreifen wollen, können Sie entsprechende Änderungen vornehmen.

Immer wenn die Option Debug zum Systemmenü hinzugefügt wird, wird die Meldungsnummer der Steuertafel mit der Option gleichgesetzt und zurückgegeben. Die Meldungsnummer wird im Parameter `wParam` der Meldung `WM_SYSCOMMAND` übergeben. Um die Steuertafel anzuzeigen, muß man diese Meldung explizit abfangen und dann die Funktion `DebugControl` aufrufen.

Der Parameter, der in `DebugSetup` die maximale Anzahl Zeilen im Fenster angibt, begrenzt den Speicherverbrauch der Utility, während im Fenster Debug-Zeilen ausgegeben werden. Das Debug-Fenster besteht aus einer verschiebbaren Listenbox, die eine Reihe von formatierten Debug-Zeilen enthält. Die Debug-Zeilen in diesem Fenster

werden laufend an die vorhandenen angehängt, bis das angegebene Maximum erreicht ist. Ab diesem Punkt ersetzt jede weitere Debug-Zeile eine frühere. Obwohl die maximale Anzahl der Debug-Zeilen, die im Fenster verwaltet werden können, nur durch den vorhandenen Speicher eingeschränkt ist, sollte man die Anzahl in den meisten Fällen auf unter 100 begrenzen.

Mit der Funktion `DebugControl` kann man die Debug-Steuertafel anzeigen und die aktuellen Debug-Optionen verändern (*Bild 2*). Diese Funktion wird normalerweise als Teil der Routine aufgerufen, die die Meldung `WM_SYSCOMMAND` behandelt, die wiederum aktiviert wird, wenn der Benutzer im Systemmenü die Option Debug wählt. Diese Funktion ruft `DebugControlDlgFn` mit der vordefinierten Dialogbox-Ressource auf.

`DebugControlDlgFn` ist für die Bearbeitung aller Meldungen zuständig, die mit der Dialogbox (der Steuertafel) von Debug zu tun haben. Der Initialisierungsabschnitt dieser Funktion (unter `WM_INITDIALOG`) setzt die aktuellen globalen Debug-Optionen in entsprechende Schalter und Editierfelder um. Der Befehlsabschnitt (unter `WM_COMMAND`) verarbeitet die verschiedenen Schaltermeldungen und aktiviert oder deaktiviert Felder dementsprechend. Der Beendigungsabschnitt (unter `BUG_OK`) liest den aktuellen Status der Steuertafel und wandelt ihn wieder zurück in die Felder der globalen Datenstruktur `Options`. Gleichzeitig analysiert er (in einer zugegebenermaßen plumpen Weise) das Filter-Editierfeld und definiert eine Liste der akzeptierten Klassifizierungscodes für Debug-Zeilen.

Schließlich erzeugt die Funktion `Debug` formatierte Debug-Zeilen. Sie wird üblicherweise von vielen Stellen der Anwendung aus aufgerufen. Der erste Parameter dieser Funktion ist der Zeilenklassifizierungscode, der von dem Filter-Abschnitt der Debug-Funktion verwendet wird. Wenn der Debug-Filter aktiviert ist, wird dieser Klassifizierungscode mit einer Liste zu filternder Kategorien verglichen. Nur Meldungen mit Codes in dieser Liste werden angezeigt oder auf Diskette geschrieben. So kann man die Meldungen ausfiltern, an denen man nicht interessiert ist.

Man kann zum Beispiel ganz willkürlich allen Zeichenereignissen eine 1, alle Zwischenablage-Ereignissen eine 2 und allen anderen eine 3 zuordnen. Wenn man dann alle Zeilen bis auf die Zwischenablage- und Zeichenereignisse ausschließen will, würde man den Filter mit der Bereichsangabe 1,2 aktivieren. Alle Zeilen, denen der Code 3 zugewiesen wurde, werden dann von der Anzeige im Fenster oder dem Protokoll in der Datei ausgeschlossen. So erhält man ein konzentriertes Ausführungsprofil, mit dem man sich auf die speziellen Bereiche eines Programms konzentrieren kann, die einen interessieren.

Der zweite Parameter der Debug-Funktion ist ein Formatsteuerstring, der den Inhalt der zu erzeugenden Zeile definiert. Dieser Steuerstring ist identisch mit dem, der in der Standard-C-Funktion `printf` verwendet wird.

```
/*
 * DebugControl( hWnd ) : BOOL FAR PASCAL
 *
 *      hWnd    main application window handle
 *
 * This function enables the user to monitor and adjust the
 * debug control panel settings.  When this function is called
 * a modal dialog box is displayed which allows the user to
 * set the debug options.
 *
 * The value returned indicates how the user terminated the
 * dialog box - TRUE implies the OK button was pressed, FALSE
 * the CANCEL button.
 *
 */

BOOL FAR PASCAL DebugControl( hWnd )
    HWND        hWnd;
{
    /* local variables */
    FARPROC lpProc;             /* temporary function address */
    BOOL        bResult;        /* boolean result value */
    HANDLE      hInstance;      /* window instance handle */

    /* retrieve instance handle */
    hInstance = GetWindowWord( hWnd, GWW_HINSTANCE );

    /* display dialog box */
    lpProc = MakeProcInstance( DebugControlDlgFn, hInstance );
    bResult = DialogBox(hInstance,(LPSTR)"DebugControl",hWnd,
                        lpProc);
    FreeProcInstance( lpProc );

    /* return final result */
    return( bResult );

}


/*
 * DebugControlDlgFn(hDlg,wMsg,wParam,lParam) : BOOL FAR PASCAL
 *
 *      hDlg        window handle
 *      wMsg        dialog message
 *      wParam      word parameter
 *      lParam      long parameter
 *
 * This function processes all the messages related to the
 * debug control panel dialog box.  With the dialog box the
 * user can add, change, and remove debugging channels.
 *
 */

BOOL FAR PASCAL DebugControlDlgFn( hDlg, wMsg, wParam, lParam )
    HWND        hDlg;
    WORD        wMsg;
    WORD        wParam;
    LONG        lParam;
{
    /* local variables */
    BOOL        bResult;        /* boolean result variable */

    /* initialization */
    bResult = TRUE;

    /* process message */
    switch( wMsg )
        {
    case WM_INITDIALOG :        /* initialize dialog box */

        {
    /* local variables */
    WORD    i;                  /* loop variable */
    int xPopup;                 /* popup x position */
    int yPopup;                 /* popup y position */
    int cxPopup;                /* width of popup window */
    int cyPopup;                /* height of popup window */
    RECT    rectWnd;            /* temporary window rect */
    char    szToken[8];         /* character token */
    char    szFilterList[64]; /* filter list string */
```

```
/* The following code centers the dialog box in the middle */
/* of the screen.  It uses the dimensions of the display */
/* and the dialog box and repositions the dialog box */
/* accordingly. */

      /* retrieve popup rectangle  */
      GetWindowRect( hDlg, (LPRECT)&rectWnd );


      /* calculate popup extents */
      cxPopup = rectWnd.right - rectWnd.left;
      cyPopup = rectWnd.bottom - rectWnd.top;


      /* calculate new location & move dialog box */
      xPopup = ( GetSystemMetrics(SM_CXSCREEN) - cxPopup) /2;
      yPopup = ( GetSystemMetrics(SM_CYSCREEN) - cyPopup) /2;

      MoveWindow( hDlg, xPopup, yPopup, cxPopup, cyPopup,
      TRUE);

      /* define filter list */
      szFilterList[0] = 0;
      if ( Options.wFilterSize ) {
          sprintf( szFilterList, "%u", Options.Filter[0] );
          for ( i=1; i<Options.wFilterSize; i++ ) {
              sprintf( szToken, ",%u", Options.Filter[i] );
              strcat( szFilterList, szToken );
          }
      }

      /* check ON-OFF radio buttons */
      CheckRadioButton(
          hDlg,
          BUG_OFF,
          BUG_ON,
          (Options.bActive) ? BUG_ON :BUG_OFF
  );


      /* define check boxes */
      CheckDlgButton( hDlg, BUG_COUNT, Options.bCount );
      CheckDlgButton( hDlg, BUG_DISPLAY, Options.bDisplay );
      CheckDlgButton( hDlg, BUG_FILTER, Options.bFilter );
      CheckDlgButton( hDlg, BUG_LOG, Options.bLog );

      /* display filter categories & log file name */
      SetDlgItemText( hDlg, BUG_FILTERLIST, szFilterList );
      SetDlgItemText( hDlg, BUG_LOGFILE, Options.szLogFile );

      /* disable check boxes & edit fields if debug
      inactive */
      if ( !Options.bActive ) {

          /* disable check boxes */
          ENABLE( hDlg, BUG_COUNT, FALSE );
          ENABLE( hDlg, BUG_DISPLAY, FALSE );
          ENABLE( hDlg, BUG_FILTER, FALSE );
          ENABLE( hDlg, BUG_LOG, FALSE );

          /* disable edit fields */
          ENABLE( hDlg, BUG_FILTERLIST, FALSE );
          ENABLE( hDlg, BUG_LOGFILE, FALSE );

      } else {

          /* enable edits field */
          ENABLE( hDlg, BUG_FILTERLIST,
              IsDlgButtonChecked(hDlg,
              BUG_FILTER) );
          ENABLE( hDlg, BUG_LOGFILE,
              IsDlgButtonChecked(hDlg,
              BUG_LOG) );

          }

      }

      break;
  case WM_COMMAND : /* dialog box command */
```

Die Parameter, die auf diesen Steuerstring folgen, werden zusammen mit dem Format-Steuerstring verwendet, entsprechend umgewandelt und ausgegeben. Hier einige Beispiele, wie die Debug-Funktion aufgerufen werden kann:

```
Debug( 1, "WM_SIZE: [%u,%u], LOWORD(lParam),
                             HIWORD(lParam) );
Debug( 2, "WM_MOVE: [%u,%u]", LOWORD(lParam),
                             HIWORD(lParam));
Debug( 3, "WM_CHAR: [%c,%u]", wParam, wParam );

Debug( 4, "WM_ACTIVATE: [%s]", (wParam)?"on":"off" );
```

Beachten Sie, wie die variable Anzahl Parameter von der Debug-Funktion gehandhabt wird. Solange Sie nicht die Größe der Struktur, die als Wert übergeben wird, ändern, sind Sie auf 64 Byte für die Parameterübergabe eingeschränkt. Sie werden vielleicht mit Alternativen zu dieser Art der Parameterübergabe experimentieren wollen. Eine Empfehlung meinerseits ist die Verwendung der C 5.x-Funktion `vprintf` mit den entsprechenden ANSI-C- oder UNIX-System V-Varianten.

Einbau der Debug-Utility

Bevor Sie die Tools der Debug-Utility verwenden können, um die interne Funktionsweise Ihrer Anwendung zu überprüfen, müssen sie mehrere Schritte ausführen:

Kopieren Sie die folgenden Dateien in das Verzeichnis, wo Sie Ihre Anwendung kompilieren:

DEBUG.H

DEBUG.C

Definieren sie eine neue Compiler-Befehlszeilenvariable in der Make-Datei Ihrer Anwendung. Diese Variable kann dann mit `#if`-Anweisungen abgefragt und so der Debug-Code erzeugt werden. Sie können zum Beispiel folgende Compiler-Flags verwenden:

```
STDFLAGS=-c -u -AS -Gsw -Os -Zep
BUGFLAGS=-c -u -AS -Gsw -Os -Zep -DDEBUG
bugtest.obj: bugtest.c
cl $(BUGFLAGS) bugtest.c
```

Ändern Sie die Make-Datei für Ihre Anwendung so, daß das Modul `DEBUG.OBJ` hinzugefügt wird, wenn Sie Ihr Programm erstellen. Ändern Sie die Make-Datei zum Beispiel von

```
bugtest.exe: bugtest.obj bugtest.def bugtest.res
    link4 bugtest /AL:16 /NOE,,,slibw,bugtest.def
```

in

```
bugtest.exe: bugtest.obj debug.obj bugtest.def bugtest.res
    link4 bugtest+debug /AL:16 /NOE,,,slibw,bugtest.def
```

Definieren Sie eine Nummer für die Debug-Kontrollmeldung in der Header-Datei Ihrer Anwendung. Diese

Meldung wird von Windows übergeben, wenn Sie im Systemmenü Debug auswählen.

Exportieren Sie `DebugControlDlgFn` indem Sie die Funktion in der DEF-Datei Ihrer Anwendung angeben. Wenn Sie das nicht tun, führt dies zu fehlerhaftem Verhalten. Sie können den Export-Abschnitt der DEF-Datei zum Beispiel von

```
EXPORTS
    TestWndFN @1
```

in

```
EXPORTS
    TestWndFN @1
    DebugControlDlgFn @2
```

ändern.

Fügen Sie die markierten Zeilen aus *Listing 6* in die Ressourcendatei Ihrer Anwendung ein.

Rufen Sie in Ihrem Programm die Funktion `DebugSetup` kurz nach der Anlage des Hauptfensters auf. Vergewissern Sie sich, daß das Hauptfenster ein Systemmenü hat und der Funktion eine gültige Handle übergeben wird. Es ist guter Programmierstil, diesen Funktionsaufruf in eine Bedingung zu stellen:

```
#if DEBUG
 DebugSetup( hMainWnd, BUG_CONTROL, 100);
#endif
```

Verwenden Sie die Anweisung `#include "debug.h"` in jeder Quelldatei, in der Sie Routinen der Debug-Utility verwenden. Die Header-Datei enthält Funktionsdefinitionen und Aufrufkonventionen, die diese Routinen benötigen.

Schreiben Sie in Ihr Programm an interessanten Stellen Debug-Anweisungen. Versuchen Sie, Ihre Debug-Anweisungen in Kategorien einzuteilen.

Beachten Sie, daß das Debuggen standardmäßig ausgeschaltet ist (OFF) und Anweisungen, die auftreten, bevor die Utility aktiviert wird, ignoriert werden. Ein Beispiel für diese Verwendung ist der Einsatz von Debug-Anweisungen, um die Verarbeitung der Meldungen für die Anlage des Fensters in der Hauptfensterfunktion Ihrer Anwendung zu überwachen.

Kompilieren und linken Sie Ihre Anwendung neu.

BUGTEST	Make-Datei
BUGTEST.C	Quellcode
BUGTEST.RC	Ressourcen-Datei
BUGTEST.ICO	Sinnbild
BUGTEST.DEF	Moduldefinitionsdatei

Tabelle 1: *Eine Liste der Dateien, die für das Beispielprogramm benötigt werden, das die Debug-Utility verwendet.*

```
/* process sub-message */
switch( wParam )
    {
case BUG_ON : /* turn debug on */

    /* redefine radio button */
    CheckRadioButton( hDlg, BUG_OFF, BUG_ON,
    BUG_ON );

    /* enable check boxes */
    ENABLE( hDlg, BUG_COUNT, TRUE );
    ENABLE( hDlg, BUG_DISPLAY, TRUE );
    ENABLE( hDlg, BUG_FILTER, TRUE );
    ENABLE( hDlg, BUG_LOG, TRUE );

    /* enable edits field */
    ENABLE( hDlg, BUG_FILTERLIST,
        IsDlgButtonChecked(hDlg, BUG_FILTER) );
    ENABLE( hDlg, BUG_LOGFILE,
        IsDlgButtonChecked(hDlg, BUG_LOG) );

    break;
case BUG_OFF : /* turn debug off */

    /* redefine radio button */
    CheckRadioButton( hDlg, BUG_OFF, BUG_ON, BUG_OFF );

    /* enable check boxes */
    ENABLE( hDlg, BUG_COUNT, FALSE );
    ENABLE( hDlg, BUG_DISPLAY, FALSE );
    ENABLE( hDlg, BUG_FILTER, FALSE );
    ENABLE( hDlg, BUG_LOG, FALSE );

    /* enable edit fields */
    ENABLE( hDlg, BUG_FILTERLIST, FALSE );
    ENABLE( hDlg, BUG_LOGFILE, FALSE );

    break;
case BUG_COUNT : /* count debug events */
    TOGGLE( hDlg, BUG_COUNT );
    break;
case BUG_DISPLAY : /* display debug events */
    TOGGLE( hDlg, BUG_DISPLAY );
    break;
case BUG_FILTER : /* filter debug events */
    TOGGLE( hDlg, BUG_FILTER );
    ENABLE( hDlg, BUG_FILTERLIST,
        IsDlgButtonChecked(hDlg,
        BUG_FILTER) );
break;
case BUG_LOG : /* log debug events */
    TOGGLE( hDlg, BUG_LOG );
    ENABLE( hDlg, BUG_LOGFILE,
        IsDlgButtonChecked(hDlg,
        BUG_LOG) );
    break;
case BUG_OK : /* done with debug control panel */

    {
        /* local variables */
        WORD    i;
        char    *spToken;
        char    szToken[8];
        char    szCaption[64];
        char    szFilterList[32];

        /* capture radio button state */
        Options.bActive =
        IsDlgButtonChecked( hDlg, BUG_ON );


    /* capture check box states */
    Options.bCount = IsDlgButtonChecked( hDlg,
    BUG_COUNT);
    Options.bDisplay=IsDlgButtonChecked(hDlg,
    BUG_DISPLAY);
    Options.bFilter=IsDlgButtonChecked
    ( hDlg,BUG_FILTER);
    Options.bLog = IsDlgButtonChecked( hDlg,
    BUG_LOG );
```

```
/* capture filter list changes - no limit
checking! */
GetDlgItemText(
    hDlg,
    BUG_FILTERLIST,
    szFilterList,
    sizeof(szFilterList)
);

Options.wFilterSize = 0;
spToken = strtok( szFilterList, " ,;" );
while ( spToken ) {
    Options.Filter
    [ Options.wFilterSize++ ] = atoi
    ( spToken );
    spToken = strtok( NULL, " ,;" );
}

/* capture log file changes */
GetDlgItemText(
    hDlg,
    BUG_LOGFILE,
    Options.szLogFile,
    sizeof(Options.szLogFile)
);

/* display listbox if necessary */
if ( Options.bActive ) {

    /* update viewport window
    caption */
    if ( Options.bFilter &&
    Options.wFilterSize ) {

        /* define filter list */
        szFilterList[0] = 0;
        if ( Options.wFilterSize ) {
            sprintf
            ( szFilterList, "%u",
            Options.Filter[0] );
            for ( i=1;
            i<Options.
            wFilterSize; i++ ) {
            sprintf( szToken,
            ",%u",
            Options.Filter[i] );
            strcat( szFilterList,
            szToken );
        }
    }

        sprintf(
            szCaption,
            "%s - (%s)",
            Options.szLogFile,
            szFilterList
        );

    } else
        sprintf( szCaption,
        "%s -                (ALL)",
        Options.szLogFile );

    /* define caption & make window
    visible */
    SetWindowText( Options.hListBox,
    szCaption );
    ShowWindow( Options.hListBox,
    SHOW_OPENWINDOW );

} else
    ShowWindow( Options.hListBox,
    HIDE_WINDOW );
    /* exit */
    EndDialog( hDlg, TRUE );

}
break;
```

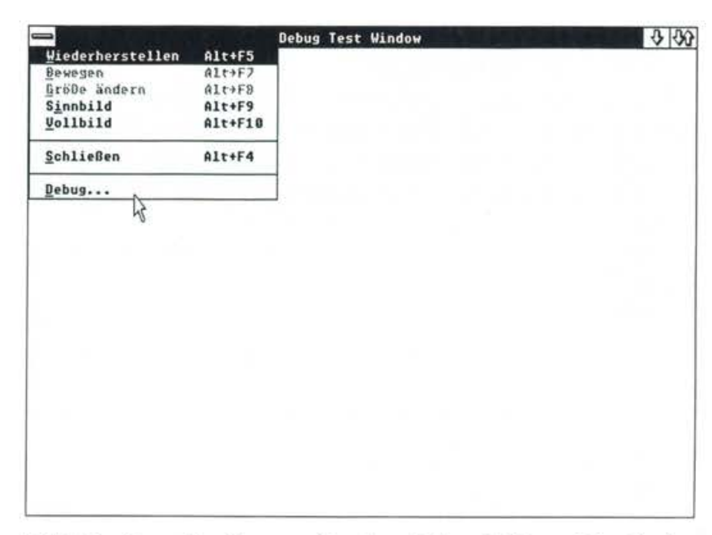


Bild 3: *Das Testfenster für die Debug-Utility. Die Option Debug ist zum Systemmenü hinzugefügt worden.*

Benutzung der Debug-Utility

Damit Sie die Debug-Utility so gut wie möglich verstehen, wollen wir uns ihren Einsatz einmal in einer Beispielanwendung wie `BUGTEST.EXE` ansehen (*Listings 3 bis 6*). In den Listings sind die Zeilen, in denen die Debug-Utility verwendet wird, hervorgehoben. Sie können diese Hervorhebungen als Vorlage für den Einbau der Utility in Ihr eigenes Programm verwenden.

Achten Sie besonders auf `TestWndFn` in `BUGTEST.C`. Diese Fensterfunktion enthält mehrere explizite Verweise auf die Debug-Utility. Zunächst sehen Sie eine Reihe von Debug-Anweisungen, die den Inhalt mehrerer Windows-Meldungen ausgeben. Sehen Sie sich an, wie die Meldungen klassifiziert werden und erinnern Sie sich daran, daß Sie Kategorien herausfiltern können, um die Generierung von Debug-Zeilen zu begrenzen.

Sie sollten sich auch die Sonderbehandlung der Meldung `BUG_CONTROL` bei der Bearbeitung der Meldung `WM_SYSCOMMAND` genau ansehen. Diese Meldung wird erzeugt, wenn Sie im Systemmenü Debug auswählen. Durch den Aufruf von `DebugControl` wird die Steuertafel von Debug angezeigt, mit der Sie das Debuggen ein- und ausschalten und verschiedene Ausgabeoptionen einstellen können.

Um `BUGTEST.EXE` zu erzeugen benötigen Sie Microsoft C 5.0 oder 5.1, Windows 2.03 und das Windows 2.03 Software Development Kit (SDK). Wenn Sie das SDK für 2.03 nicht haben, können Sie das Programm (mit eventuellen kleinen Korrekturen) auch mit dem 1.04 SDK zum Laufen bringen. Das Sinnbild `BUGTEST.ICO` ist nicht abgebildet, Sie können sich jedoch ein beliebiges Sinnbild mit IconEdit selbst erstellen. (Alle Dateien für `DEBUG` und `BUGTEST` sind wie immer auch auf der Microsoft System Journal-Diskette erhältlich.)

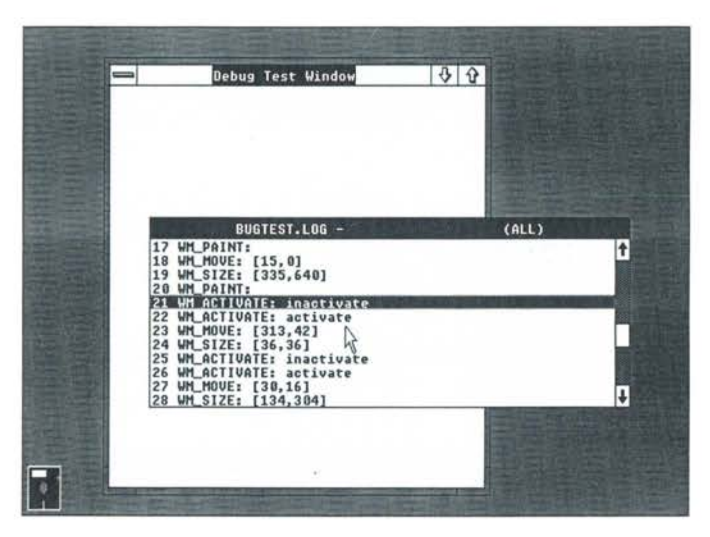


Bild 4: *Die Listenbox von BUGTEST.*

Wenn Sie die Anwendung BugTest erstellt haben, sollten Sie ein wenig mit der Debug-Utility herumexperimentieren. Wenn Sie das Programm zum ersten Mal starten, erscheint ein Standardfenster. Mit der Maus oder der Tastatur können Sie dieses Fenster verschieben, seine Größe verändern oder es in ein Sinnbild verwandeln, genauso wie bei jedem anderen Fenster (*Bild 3*). Immer wenn Sie eine dieser Operationen ausführen, werden zahlreiche Debug-Zeilen erzeugt, sie werden von der Debug-Utility jedoch ignoriert, bis Sie das Debuggen mit der Steuertafel explizit einschalten (ON).

Um auf die Steuertafel zuzugreifen, wählen Sie Debug im Systemmenü. In dieser Dialogbox können sie das Debuggen einschalten und Optionen angeben. Sie können die Optionen Count und Display aktivieren, wodurch die Debug-Zeilen numeriert und die Ergebnisse in der Listenbox angezeigt werden (*Bild 4*). In dieser Listenbox können Sie sich die Debugzeilen ansehen, während Sie von der Hauptanwendung erzeugt werden (*Bild 5*).

Denken Sie daran, daß Sie mit der Filter-Option bestimmte Kategorien von Debug-Zeilen auswählen können. Beachten Sie auch, daß diese Version der Debug-Utility auf 32 Filter-Kategorien eingestellt ist. Wenn dies nicht ausreicht, können Sie einfach die Definition von `MAX_FILTER` in `DEBUG.C` auf einen größeren Wert abändern und die Utility neu kompilieren.

Die Option Log-File funktioniert ähnlich wie die Display-Option, nur werden die Debug-Zeilen dann an die Datei angehängt, die Sie angeben. Sie können sich die Debug-Zeilen gleichzeitig anzeigen und in der Datei protokollieren lassen, das System wird dadurch jedoch erheblich langsamer. Wenn ich Protokoll-Dateien verwende, füge ich folgende Zeile in den Extension-Abschnitt von `WIN.INI` ein:

```
LOG=NOTIZ.EXE ^.LOG
```

```
        case BUG_CANCEL :   /* cancel control panel */
        case IDCANCEL :
            EndDialog( hDlg, FALSE );
            break;
        default :   /* ignore all others */
            bResult = FALSE;
            break;
        }

        break;
    default :
        bResult = FALSE;
        break;
    }

    /* return final result */
    return( bResult );

}

/*
 * Debug( wCategory, ParmList) : BOOL FAR
 *
 *      wCategory       debug message classification
 *      ParmList        debug parameter list
 *
 * This function outputs a formatted message to the debug
 * utility routines.  The wCategory variable determines how
 * the message is filtered.  The ParmList is a standard printf
 * parameter list. A value of TRUE is returned if the message
 * was successfully processed and not filtered.
 *
 * Note that this function CANNOT be declared of type PASCAL
 * as it uses a variable number of arguments! Also note that
 * alternative methods are available for handling a variable
 * number of function parameters.
 *
 */

BOOL FAR Debug( wCategory, ParmList )
    WORD        wCategory;
    struct {
        char    cBytes[64];
    } ParmList;
{
    /* local variables */
    WORD    i;                  /* temporary loop variable */
    int     hFile;              /* log file handle */
    BOOL    bResult;            /* result of function */
    BOOL    bInclude;           /* include message flag */
    char    szStatement[132]; /* temporary statement */

    /* initialization */
    bResult = FALSE;

    /* process statement if debug active */
    if ( Options.bSetup && Options.bActive ) {

        bInclude = TRUE;

        /* check for inclusion in filter list */
        if ( Options.bFilter ) {

            /* search filter list */
            for (
                i=0;
                (i<Options.wFilterSize)&&
                (Options.Filter[i]!=
                wCategory);
                i++
            );

            /* turn off include if not found */
            if ( i >= Options.wFilterSize )
                bInclude = FALSE;

        }

        /* format debug statement if included */
        if ( bInclude ) {
```

```
        /* prepare for debug statement */
        i = 0;
        bResult = TRUE;
        Options.lMsgCount++;

        /* define debug statement */
        if ( Options.bCount )
            i = sprintf( szStatement, "%ld ",
            Options.lMsgCount );
        i += sprintf( &szStatement[i], ParmList );

        /* display debug statement in listbox */
        if ( Options.bDisplay ) {


            /* Note: since we are passing
             * the formatted debugging
             * statement to the listbox using
             * a SendMessage we cannot do
             * anything which would cause
             * memory to be shuffled - this
             * invalidates the pointer!
             */

            /* add new statement to listbox */
            SendMessage(
                Options.hListBox,
                LB_ADDSTRING,
                (WORD)0,
                (LONG)(LPSTR)szStatement
            );

            /* remove first statement from
            listbox if necessary */
            if ( Options.wEntries >=
            Options.wMaxEntries ) {
                SendMessage(
                    Options.hListBox,
                    LB_DELETESTRING,
                    0,
                    (LONG)0
                );
            } else
                Options.wEntries++;

            /* make last statement visible */
            SendMessage(
                Options.hListBox,
                LB_SETCURSEL,
                (WORD)Options.wEntries-1,
                (LONG)0
            );

        }

        /* output debug statement to logfile */
        if ( Options.bLog ) {

            /* open or create log file */
            hFile = open( Options.szLogFile,
            MODE_APPEND, SHARED);
            if ( hFile < 0 )
            hFile=open(Options.szLogFile,
            MODE_CREATE,SHARED);

            /* write message to file if
               sucessful */
            if ( hFile > 0 ) {
                write( hFile, szStatement, i );
                write( hFile, "\r\n", 2 );
                close( hFile );
            }
        }
    }
  }
  /* return result */
  return( bResult );
}
```

Listing 2: *Die Debug-Utility.*

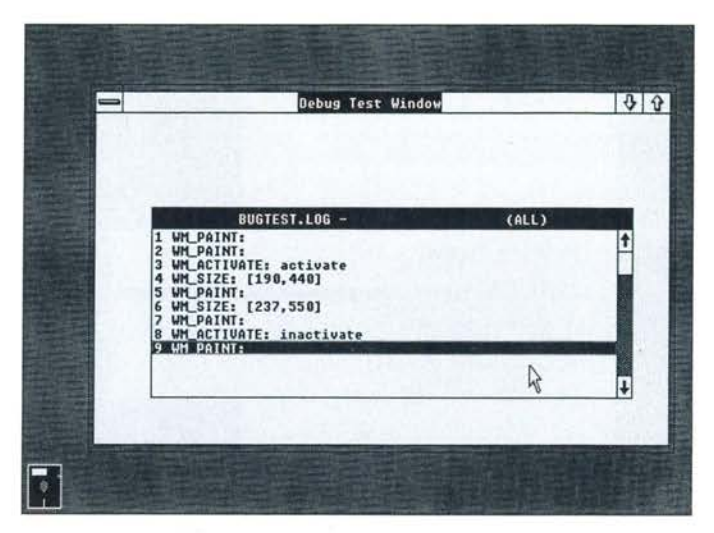


Bild 5: *In der Listenbox werden die Debug-Zeilen angezeigt.*

Dadurch kann ich mir eine Protokolldatei mit einem Doppelklick auf ihren Namen vom Notizbuch anzeigen lassen. Denken Sie aber daran, daß dieses Programm keine großen Protokolldateien laden kann, doch das sollte eigentlich kein großes Problem sein.

Die Debug-Utility wird sicherlich nicht solche Programme wie Windows-CodeView verdrängen. Doch für einfaches Debuggen bietet sie viel Übersicht über die Funktionsweise eines Programms und dürfte deshalb eine nützliche Erweiterung der Toolbox jedes Programmierers sein.

Mit der Debug-Utility können eine ganze Reihe von Windows-Problemen gelöst werden. Mit den Listings auf diesen Seiten als Ausgangspunkt können Sie diese Utility in vielen interessanten Bereichen ausbauen. Sie können sich in das System einklinken, um eine allgemeine Meldungsüberwachung durchzuführen und so Probleme zwischen Anwendungen aufzuspüren. Sie können auch mit mehreren Debug-Fenstern für eine Anwendung experimentieren. Oder Sie könnten die Utility in eine dynamische Linkbibliothek bringen und mehrere Anwendungen gleichzeitig testen. Diese und hoffentlich viele weitere kreative Erweiterungen überlasse ich Ihnen.

Kevin P. Welch

```
STDFLAGS=-c -u -AS -Gsw -Os -Zep
BUGFLAGS=-c -u -AS -Gsw -Os -Zep -DDEBUG

bugtest.res: bugtest.rc bugtest.ico
    rc -r bugtest.rc

debug.obj: debug.h debug.c
    cl $(STDFLAGS) debug.c

bugtest.obj: bugtest.c
    cl $(BUGFLAGS) bugtest.c

bugtest.exe: bugtest.obj bugtest.def bugtest.res debug.obj
    link4 bugtest+debug /AL:16 /NOE,,,slibw slibcew,bugtest.def
    rc bugtest.res
```

Listing 3: *Die Make-Datei BUGTEST.*

```
/*
 * DEBUG TEST PROGRAM - SOURCE CODE
 *
 * LANGUAGE : Microsoft C 5.0
 * MODEL    : small
 * STATUS   : operational
 *
 * 12/11/87 1.00 - Kevin P. Welch - initial creation.
 *
 */

#include <windows.h>
#include "debug.h"

/* local definitions */
#define BUG_CONTROL 201

/* function definitions */
LONG FAR PASCAL TestWndFn( HWND, WORD, WORD, LONG );


/*
 * MAINLINE - BUG TEST PROGRAM
 *
 * This mainline initializes the test program and processes
 * and dispatches all messages relating to the debug test
 * window.
 *
 */

int PASCAL WinMain( hInstance, hPrevInstance, lpsCmd, wCmdShow
)
    HANDLE  hInstance;
    HANDLE  hPrevInstance;
    LPSTR   lpsCmd;
    WORD    wCmdShow;
{
    /* local variables */
    MSG Msg;    /* current system message */

    /* initialization */
    if (TestInit(hInstance, hPrevInstance, lpsCmd, wCmdShow ))
{

        /* process system messages until finished */
        while ( GetMessage( (LPMSG)&Msg, NULL, 0, 0 ) ) {
            TranslateMessage( (LPMSG)&Msg );
            DispatchMessage( (LPMSG)&Msg );
        }

        /* terminate application */
        exit( Msg.wParam );

    } else
        exit( FALSE );

}


/*
 *  TestInit(hInstance,hPrevInstance,lpsCmd,wCmdShow) : BOOL;
 *
 *      hInstance       current instance handle
 *      hPrevInstance   handle to previous instance
 *      lpsCmd          command line string
 *      wCmdShow        window display flag
 *
 * This utility function performs all the initialization
 * required for testing the debug utility.  Included in this
 * program is the registry and creation of the main window &
 * the installation of the debug utility code.
 *
 */

static BOOL TestInit(hInstance,hPrevInstance,lpsCmd,wCmdShow)
    HANDLE      hInstance;
    HANDLE      hPrevInstance;
    LPSTR       lpsCmd;
    WORD        wCmdShow;
{
    /* local variables */
    HWND        hWnd;       /* current window handle */
    BOOL        bResult;    /* result of initialization */
    WNDCLASS    WndClass;   /* window class */

    /* initialization */
    bResult = FALSE;

    /* register window class */
    if ( !hPrevInstance ) {

        /* define window class */
        memset( &WndClass, 0, sizeof(WNDCLASS) );
        WndClass.lpszClassName = (LPSTR)"TestWindow";
        WndClass.hCursor = LoadCursor( NULL, IDC_ARROW );
        WndClass.lpszMenuName = (LPSTR)NULL;
        WndClass.style = CS_HREDRAW | CS_VREDRAW;
        WndClass.lpfnWndProc = TestWndFn;
        WndClass.hInstance = hInstance;
        WndClass.hIcon = LoadIcon( hInstance, "BugTestIcon" );
        WndClass.hbrBackground = (HBRUSH)(COLOR_MENU + 1);

        /* register maze window class */
        if ( RegisterClass( (LPWNDCLASS)&WndClass ) ) {


            /* create window */
            hWnd = CreateWindow(
                "TestWindow",           /* class name */
                "Debug Test Window",/* caption */
                WS_TILEDWINDOW,         /* style */
                CW_USEDEFAULT,          /* x position */
                CW_USEDEFAULT,          /* y position */
                CW_USEDEFAULT,          /* width */
                CW_USEDEFAULT,          /* height */
                (HWND)NULL,             /* parent window */
                (HMENU)NULL,            /* menu */
                hInstance,              /* application */
                (LPSTR)NULL             /* other data */
                );

            /* continue if successful */
            if ( hWnd ) {

            /* Here is where the debug utility is
             * installed into the program. A response
             * message number is provided along with the
             * maximum number of debug statements which
             * will be maintained by the listbox. The
             * larger this number, the less global memory
             * available for your application.
             */
#if DEBUG
            DebugSetup( hWnd, BUG_CONTROL, 100 );
#endif
            /* make window visible */
            bResult = TRUE;
            ShowWindow( hWnd, wCmdShow );

            }

        }

    }

    /* return result */
    return( bResult );

}
```

```
/*
 * TEST WINDOW MESSAGE PROCESSING PROCEDURE
 *
 * TestWndFn(hWnd, wMessage, wParam, lParam ) : LONG FAR PASCAL
 *
 *      hWnd        window handle
 *      wMessage    message number
 *      wParam      additional message information
 *      lParam      additional message information
 *
 * This window function processes all the messages related to
 * the debug test window.  Using the system menu the user can
 * display the debug control panel dialog box.
 *
 */

LONG FAR PASCAL TestWndFn( hWnd, wMessage, wParam, lParam )
    HWND        hWnd;
    WORD        wMessage;
    WORD        wParam;
    LONG        lParam;
{
    /* local variables */
    LONG        lResult;    /* result of message */

    /* initialization */
    lResult = FALSE;

#if DEBUG
    /* sample debugging output */
    switch( wMessage )
        {
    case WM_MOVE :
        Debug( 1, "WM_MOVE: [%u,%u]", HIWORD(lParam),
        LOWORD(lParam) );
        break;
    case WM_SIZE :
        Debug( 1, "WM_SIZE: [%u,%u]", HIWORD(lParam),
        LOWORD(lParam) );
        break;
    case WM_CHAR :
        Debug( 2, "WM_CHAR: [%c,%u]", wParam, wParam );
        break;
    case WM_ACTIVATE :
        Debug( 3, "WM_ACTIVATE: %s",
        (wParam)?"activate":"inactivate" );
        break;
    case WM_ACTIVATEAPP :
        Debug( 3, "WM_ACTIVATEAPP: %s",
        (wParam)?"activate":"inactivate" );
        break;
    case WM_PAINT :
        Debug( 4, "WM_PAINT:" );
        break;
    default :
        break;
    }
#endif

    /* process each message */
    switch( wMessage )
        {
    case WM_SYSCOMMAND : /* system command */

        /* In here you need to handle the special case where
         * the user asks for the debug control panel to be
         * displayed.
         * To do so you need to trap the control panel response
         * message you provided when installing the debug
         * utility.
         */

        /* process sub-message */
        switch( wParam )
            {
#if DEBUG
        case BUG_CONTROL : /* debug control panel */
            DebugControl( hWnd );
            break;
#endif
        default :
            lResult = DefWindowProc( hWnd, wMessage, wParam,
            lParam );
            break;
        }

        break;
    case WM_DESTROY :   /* destroy window */
        PostQuitMessage( 0 );
        break;
    default : /* send to default */
        lResult = DefWindowProc( hWnd, wMessage, wParam,
        lParam );
        break;
    }
    /* return normal result */
    return( lResult );
}
```

Listing 4: *Das Beispielprogramm BUGTEST.C.*

```
NAME        BUGTEST

DESCRIPTION 'Debug Test Utility'

STUB        'WINSTUB.EXE'

CODE        MOVEABLE
DATA        MOVEABLE MULTIPLE

HEAPSIZE    4096
STACKSIZE   4096

EXPORTS
    TestWndFn           @1
    DebugControlDlgFn   @2
```

Listing 5: *Die Moduldefinitionsdatei BUGTEST.DEF.*

```
/*
 * DEBUG TEST PROGRAM - RESOURCE FILE
 *
 * LANGUAGE : Microsoft C 5.0
 * MODEL    : small
 * STATUS   : operational
 *
 * 12/11/87 1.00 - Kevin P. Welch - initial creation.
 *
 */

#include <style.h>
#include "debug.h"

BugTestIcon     ICON    bugtest.ico
```

```
DebugControl DIALOG LOADONCALL MOVEABLE DISCARDABLE 8,20,185,81
CAPTION "Debug Control Panel"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "OFF - debug &inactive."
            BUG_OFF, "button", STD_RADIO, 5, 3, 130, 12
    CONTROL "ON - debug &active."
            BUG_ON, "button", STD_RADIO, 5, 15, 128, 12
    CONTROL "&Count debug events."
            BUG_COUNT, "button", STD_CHECKBOX, 18, 26, 114, 12
    CONTROL "&Display in debug window."
            BUG_DISPLAY, "button", STD_CHECKBOX, 18, 38, 114,12
    CONTROL "&Filter"
            BUG_FILTER, "button", STD_CHECKBOX, 18, 50, 34, 12
    CONTROL ""
          BUG_FILTERLIST, "edit", STD_EDITFIELD, 55, 50, 78, 12
    CONTROL "&Log to"
            BUG_LOG, "button", STD_CHECKBOX, 18, 62, 36, 12
    CONTROL ""
            BUG_LOGFILE, "edit", STD_EDITFIELD , 55, 63, 78, 12
    CONTROL ""
            -1, "static", STD_FRAME, 142, 0, 1, 81
    CONTROL "Ok"
            BUG_OK, "button", DEF_BUTTON, 148, 4, 32, 14
    CONTROL "Cancel"
            BUG_CANCEL, "button", STD_BUTTON, 148, 21, 32, 14
END
```

Listing 6: *Die Ressourcendatei BUGTEST.RC.*

Inserentenverzeichnis

BKS Software	35
BSP Krug	27
Dr.-Ing. P. Christiani	57
ESM	27
IWT Verlag	37, 91
Mannesmann Tally	21
Markt & Technik Buchverlag	77, 92
Microsoft	40/41, 58/59
Nokia Data	2
Star Division	11
te-wi Verlag	17
Sybex Verlag	25

Termine ... Termine

Microsoft Excel für Programmierer

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrung in einer höheren Programmiersprache haben.

Die Teilnehmer lernen im Vortrag und in praktischen Übungen am PC das Konzept und die Möglichkeiten, mit Microsoft Excel-Makros Applikationen zu erstellen.

Ort	Datum	Veranstalter
Düsseldorf	19./20./21.09.	OBZ
	17./18./19.10.	OBZ
	28./29./30.11.	OBZ
München	10./11./12.10.	Integrata
Münster	12./13./14.12.	Integrata
Baden	12./13./14.09.	Integrata
Genf	05./06./07.12.	Electro Calcul
Lausanne	24./25./26.10.	Electro Calcul
Wien	12./13./14.09.	Ueberreuter
	07./08./09.11.	Ueberreuter
	05./06./07.12.	Ueberreuter

Microsoft Presentation Manager für Windows Programmierer

Ort	Datum	Veranstalter
Düsseldorf	17.10.88	OBZ
	28.11.88	OBZ
	19.12.88	OBZ
Wien	20.10.88	Ueberreuter
	21.11.88	Ueberreuter

Termine ... Termine ... Termine ... Termine

Mit Microsoft-Seminaren sicher in die Zukunft

Das Betriebssystem der Zukunft heißt Microsoft OS/2. Microsoft Windows und der Presentation Manager sind die Benutzeroberflächen der Zukunft sein. Für professionelle Entwickler bedeutet das, sich ab sofort mit dieser neuen Software auseinandersetzen zu müssen. Damit schaffen sie die Voraussetzung, schnellstmöglich Programme in der »neuen Welt« verfügbar zu haben.

Natürlich wird die Umstellung auf das neue Betriebssystem sowie die Programmerstellung nicht von heute auf morgen vollzogen sein. Um den Anfang jedoch so einfach wie möglich zu gestalten, bietet Microsoft eine Dienstleistung an: Das Microsoft Institut.

Die Spezialseminare des Microsoft Instituts vermitteln in kleinen Gruppen intensiv all das, was zum Einstieg in die Programmentwicklung nötig ist. Modernste Trainingsmethoden sowie PC-Demonstrationen und -Übungen sind selbstverständlich. Die Dozenten befassen sich auch im persönlichen Gespräch ausführlich mit den individuellen Forderungen und Problemen der Teilnehmer. So bekommen professionelle Entwickler durch professionelle Schulung die Möglichkeit, ihren hohen Wissenstand den neuen Gegebenheiten anzupassen.

Jeder Interessent in der Bundesrepublik Deutschland, der Schweiz und Österreich hat die Chance, sich mit der neuen Welt von Microsoft OS/2 und Microsoft Windows auseinanderzusetzen. Denn das Microsoft Institut arbeitet vor Ort mit kompetenten Schulungsunternehmen zusammen:

Digicomp AG, Zürich
Elektro-Calcul PI S.A., Lausanne
Integrata GmbH, Tübingen
INTEL Semiconductor GmbH, München
Olivetti Bildungs-Zentrum GmbH, Düsseldorf
Ueberreuter Media GmbH, Wien.

Die Dozenten werden speziell von Microsoft ausgebildet und stehen in ständigem Kontakt mit uns. So gibt es keine Informationsverluste: Das Wissen wird immer aktuell und aus erster Hand vermittelt. Microsoft erstellt die Seminare und die Seminarunterlagen und gewährleistet Qualität durch die Auswertung der Seminare.

Das Microsoft OS/2 Einführungs-Seminar

Das zweitägige Seminar wendet sich an PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Die Teilnehmer lernen im Vortrag und in Diskussionen das Konzept von Microsoft OS/2 kennen und erhalten einen Überblick über die Fähigkeiten und Programmierschnittstellen dieses Betriebssystems. Während des Seminars haben die Teilnehmer die Möglichkeit, das Gelernte anhand von Übungsaufgaben für sich selbst zu überprüfen.

Ort	Datum	Veranstalter
Düsseldorf	05./06.09.	OBZ
	03./04.10.	OBZ
	07./08.11.	OBZ
	05./06.12.	OBZ
Frankfurt	05./06.09.	Integrata
	14./15.11.	Integrata
	05./06.12.	Integrata
Hamburg	03./04.10.	Integrata
	12./13.12.	Integrata
München	12./13.09.	Intel
	15./16.09.	Integrata
	24./25.10.	Integrata
	10./11.11.	Integrata
	19./20.12.	Integrata
Münster	19./20.09.	Integrata
	28./29.11.	Integrata
Tübingen	26./27.09.	Integrata
	10./11.10.	Integrata
	07./08.11.	Integrata
	21./22.11.	Integrata
Graz	15./16.09.	Ueberreuter
Innsbruck	19./20.09.	Ueberreuter
Linz	07./08.11.	Ueberreuter
Salzburg	13./14.10.	Ueberreuter
Wien	15./16.09.	Ueberreuter
	19./20.09.	Ueberreuter
	05./06.10.	Ueberreuter
	29./30.11.	Ueberreuter
Genf	14./15.09.	Electro Calcul
Lausanne	14./15.11.	Electro Calcul
Zürich	21./22.09.	Digicomp
	13./14.10.	Digicomp
	01./02.12.	Digicomp
	20./21.12.	Digicomp

Der Microsoft OS/2 Workshop

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das MS-OS/2-Einführungsseminar besucht haben.

Die Teilnehmer lernen im Vortrag und praktischen Übungen am PC Family-API-Programme zu schreiben und

Termine ... Termine ... Termine ... Termine

Device-I/O-Routinen zu erstellen sowie Multitasking-Funktionen zu nutzen und eigene Dynamic-Link-Bibliotheken zu erstellen; außerdem können sie die erweiterten Speicherverwaltungsmöglichkeiten des Intel 80286 nutzen und mit Hilfe des MS-OS/2 Memory Managers programmieren. Dieses Seminar ist übrigens nicht im SDK-Preis enthalten.

Ort	Datum	Veranstalter
Düsseldorf	07./08./09.09.	OBZ
	05./06./07.10.	OBZ
	09./10./11.11.	OBZ
	07./08./09.12.	OBZ
Frankfurt	07./08./09.09.	Integrata
	07./08./09.12.	Integrata
Hamburg	05./06./07.10.	Integrata
	14./15./16.12.	Integrata
München	31./01./02.09.	Integrata
	14./15./16.09.	Intel
	21./22./23.09.	OBZ
	26./27./28.10.	Integrata
	21./22./23.12.	Integrata
Münster	21./22./23.09.	Integrata
	30./01./02.12.	Integrata
Tübingen	28./29./30.09.	Integrata
	09./10./11.11.	Integrata
Genf	26./27./28.09.	Electro Calcul
Lausanne	28./29./30.11.	Electro Calcul
Wien	05./06./07.09.	Ueberreuter
	17./18./19.10.	Ueberreuter
	14./15./16.12.	Ueberreuter
Zürich	05./06./07.10.	Digicomp
	25./26./27.10.	Digicomp
	12./13./14.12.	Digicomp

Das Microsoft Windows Einführungs-Seminar

Das zweitägige Seminar wendet sich an PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Die Teilnehmer lernen im Vortrag und in Diskussionen das Konzept von Microsoft Windows kennen und erhalten einen Überblick über dessen Fähigkeiten und Programmierschnittstellen. Dieses Seminar ist nicht im SDK-Preis enthalten.

Ort	Datum	Veranstalter
Düsseldorf	12./13.09.	OBZ
	10./11.10.	OBZ
	21./22.11.	OBZ
	12./13.12.	OBZ
Frankfurt	06./07.10.	Integrata
Hamburg	12./13.09.	Integrata
	10./11.10.	Integrata
	24./25.10.	Integrata
München	01./02.09.	Integrata
	26./27.09.	Integrata
	13./14.10.	Integrata
München	14./15.11.	Integrata
	05./06.12.	Integrata
	19./20.12.	Integrata
Münster	21./22.11.	Integrata
Tübingen	19./20.09.	Integrata
	20./21.10.	Integrata
	03./04.11.	Integrata
	28./29.11.	Integrata
	21./22.12.	Integrata
Graz	26./27.09.	Ueberreuter
Innsbruck	05./06.09.	Ueberreuter
Lausanne	03./04.10.	Electro Calcul
Linz	17./18.10.	Ueberreuter
Salzburg	28./29.11.	Ueberreuter
Wien	03./04.10.	Ueberreuter
	14./15.11.	Ueberreuter
Zürich	21./22.09.	Digicomp
	10./11.11.	Digicomp
	05./06.12.	Digicomp

Der Microsoft Windows Workshop

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das Microsoft Windows Einführungsseminar besucht haben.

Die Teilnehmer lernen im Vortrag und praktischen Übungen am PC Benutzerschnittstellen zu erstellen, die grafische Programmierschnittstelle zu nutzen, die Routinen zum Memory Management anzuwenden und dynamische Bibliotheken zu erstellen und zu benutzen. Dieses Seminar ist nicht im SDK-Preis enthalten.

Ort	Datum	Veranstalter
Düsseldorf	14./15./16.09.	OBZ
	12./13./14.10.	OBZ
	23./24./25.11.	OBZ
	14./15./16.12.	OBZ
Frankfurt	14./15./16.12.	Integrata
Hamburg	14./15./16.09.	Integrata
	26./27./28.10.	Integrata
München	07./08./09.12.	Integrata
Münster	23./24./25.11.	Integrata
Tübingen	21./22./23.09.	Integrata
	30./01./02.12.	Integrata
Lausanne	05./06./07.10.	Electro Calcul
Wien	10./11./12.10.	Ueberreuter
Zürich	26./27./28.09.	Digicomp
	28./29./30.11.	Digicomp

Fragen & Antworten zu Windows

Abfangen von Prozeduraufrufen

F: *Windows scheint die Funktionsargumente nicht auf Gültigkeit zu überprüfen. Gibt es einen Weg, dies zu ändern. Gibt es eine transparente Möglichkeit um die globale Speicherverwaltung um das Auslagern von Daten zu erweitern?*

A: Die Antwort auf beide Fragen lautet »Ja«. Es gibt eine Technik zur Aliasierung von Prozedurnamen in Windows, mit der man sich transparent in Prozeduraufrufe einklinken kann. Sie kann für jede Art von Überwachung eingesetzt werden.

Bauen Sie zunächst eine ausführbare Windows-Code-Bibliothek auf, die wie folgt aussieht:

```
HANDLE MyGlobalAlloc( flags, size )
WORD  flags;
DWORD size;
{
   /* Führen Sie hier alle gewünschten Kontrollen aus, z.B.:

        if ( !size )
            return Ø;

    */
    return RealGlobalAlloc( flags, size );
}

HANDLE FAR PASCAL main( argc, argv )
WORD argc;
LPSTR argv;
{
    return 1;
}
```

Schreiben Sie dann folgende Definitionsdatei für die obige Bibliothek:

```
FOO.DEF:

LIBRARY FOO
DESCRIPTION "Procedure call interception library."
DATA SINGLE MOVEABLE
CODE MOVEABLE DISCARDABLE
EXPORTS
        GlobalAlloc=MyGlobalAlloc
IMPORTS
         RealGlobalAlloc=KERNEL.15
```

Die laufende Nummer der Kernel-Routinen (oder jeder anderen Routine) kann festgestellt werden, indem `LIB.EXE` auf die Bibliotheken `SLIBW` oder `MLIBW` angewendet wird.

Sie bauen dann eine dynamische Link-Library auf, indem Sie IMPLIB wie folgt über `FOO.DEF` laufen lassen:

```
IMPLIB FOO.DEF FOO.LIB
```

Bauen Sie die ausführbare Bibliothek mit folgendem Befehl auf:

```
link4 foo,,,mlibw mlibc,foo.def;
```

Sie können dann eine dynamische Verbindung zu `FOO.EXE` aufbauen, indem Sie wie folgt mit `FOO.LIB` linken:

```
link4 bar,,,foo mlibw mlibc,bar.def;
```

Alle Aufrufe von `GlobalAlloc` werden dann zunächst nach `MyGlobalAlloc` umgeleitet, weil der Name gleichgesetzt wurde.

Diese Technik ist sehr leistungsfähig und kann für jede Art von Überwachungsfunktion verwendet werden. Im Fall von `GlobalAlloc` ist die Überwachung transparent in dem Sinn, daß Sie die Anwendung nicht neu kompilieren müssen, um die Überprüfung zu entfernen. Linken Sie einfach mit `MLIBW.LIB` statt mit `FOO.LIB`.

Pufferüberlauf bei WriteComm

F: *Wenn ich* `LPT1` *öffne habe ich Probleme mit der Routine* `WriteComm`. *Wenn die Übertragungsqueue beim Aufruf von* `WriteComm` *Daten enthält, gehen die Daten in der Queue verloren. Wie kann ich dieses Problem vermeiden?*

A: Nach dem Öffnen eines Kommunikationsgeräts und der passenden Einstellung schreibt man Daten mit `TransmitCommChar` zu dieser Einheit. Die Gefahr, daß Sie Daten verlieren, besteht, wenn sie die Übertragungsqueue nicht darauf überprüfen, ob diese genug Platz für die Übertragung hat. Der folgende Programmausschnitt ist ein Beispiel dafür, wie der Code, der dafür sorgt, daß dies nicht passiert, aussehen kann (abhängig davon, wie zeitkritisch Ihr Programm ist).

Überprüfen Sie immer erst die Ausgabequeue, um sicherzustellen, daß genügend Platz vorhanden ist, bevor Sie `WriteComm` verwenden. Die Ausgabequeue, die Sie mit `OpenComm` deklarieren, sollte mindestens so groß sein, wie der größte Ausgabestring.

```
nLPT1Cid = OpenComm((LPSTR) "LPT1", nInQ, nOutQ);
.
. /* Verwenden Sie SetCommState wenn die Standardein- */
. /* stellungen geändert werden sollen                */
.

/* Rufen Sie GetCommError auf, um sicherzustellen, daß kein
* Druckerfehler aufgetreten ist und wenn doch, um ihn fest-
* zustellen. Wenn es ein Problem gegeben hat, korrigieren
* Sie es. Der Aufruf von GetCommError übergibt auch die
* Anzahl Zeichen in der Queue und ermöglicht so, sicherzu-
* stellen, daß genug Platz im Puffer ist für einen Aufruf
* von WriteComm ohne Fehler.
*
* Die do-Schleife wartet, bis alle Druckerfehler korrigiert
* sind (oder beendet wenn nötig); dann wird sichergestellt,
* daß genügend Platz in der Queue vorhanden ist, um die
* gewünschte Anzahl Zeichen zu übertragen.
*/
```

```
do
  {
  while((nComErr=GetCommError(nLPT1Cid,lpLPT1COMSTAT)) != NULL )
     {
     if (nQuit = LPT1Error(nComErr))
       break;
     }
   if (nQuit)
     break;
  }
while (nOutQ - nCharToSend < lpLPT1COMSTAT->cbOutQue)
  {
  WriteComm( nLPT1Cid, nCharToSendBuf, nCharToSend);
  .
  .
  .
```

Die Anzahl der Funktionstasten feststellen

F: *Wie kann ich die von einem OEM unterstützten Tasten feststellen, z.B. die Funktionstasten F11 bis F16 und auch andere Tasten, die von verschiedenen OEMs unterstützt werden.*

A: Die folgende Beispielanwendung zeigt, wie diese Information und auch die Anzahl der Mausknöpfe festgestellt werden kann:

INFO.MAK:

```
info.obj: info.c info.mak
    cl -d -c -AS -Gsw -Od -Zped info.c

info.exe: info.obj  info.def
    link4 info, /align:16, /map/li, slibw, info.def
    mapsym info
```

INFO.DEF:

```
NAME Info

DESCRIPTION 'Get # of F keys and # of mouse buttons.'

STUB    'WINSTUB.EXE'

CODE    MOVEABLE
DATA    MOVEABLE MULTIPLE

HEAPSIZE  4096
STACKSIZE  4096

IMPORTS
    KeyboardInquire = Keyboard.1
    MouseInquire    = Mouse.1
```

INFO.C:

```
#include "windows.h"

extern WORD FAR PASCAL KeyboardInquire ( LPSTR );
extern WORD FAR PASCAL MouseInquire ( FARPROC );

typedef struct {
  char Begin_First_range;
  char End_First_range;
  char Begin_Second_range;
  char End_Second_range;
  WORD kbStateSize; /* #bytes of state info from ToAscii.     */
  WORD kbNumFuncKeys;/* No of function keys on the keyboard. */
  WORD kbHasBreak; /* true = keyboard supplies make and break*/
  WORD kbRate;      /* Maximum rate of keyboard input events. */
  } KBINFO;

KBINFO  keybdInfo;

typedef struct {
    char  Exists;
    char  Relative;
    short NumButtons;
    short Rate;
    short XThresh;
    short YThresh;
    short XRes;
    short YRes;
} MOUSEINFO;

MOUSEINFO mouseInfo;


int PASCAL WinMain( hInstance, hPrevInstance,
                    lpszCommandLine, cmdShow)
HANDLE  hInstance;
HANDLE  hPrevInstance;
LPSTR   lpszCommandLine;
int     cmdShow;

{
    register WORD  i,k;
    char buff[45];

  /* Get keyboard driver information. */
  if ((i=KeyboardInquire((LPSTR)&keybdInfo))
       != sizeof( KBINFO ))
  {
      MessageBox(NULL,(LPSTR)"Keyboard info",
                      (LPSTR)"error", MB_OK);
      return (1);
  }
  sprintf(buff, "# of Function Keys: %d",
         keybdInfo.kbNumFuncKeys );
  MessageBox (NULL,(LPSTR)buff, (LPSTR)"Keyboard", MB_OK);

  /* Get Mouse driver information. */
  if ((i=MouseInquire((FARPROC)&mouseInfo))
       != sizeof( MOUSEINFO ) ) {
      MessageBox (NULL,(LPSTR)"Mouse info",
                       (LPSTR)"error", MB_OK);
      return (1);
  }
  sprintf(buff,"# of mouse buttons: %d",mouseInfo.NumButtons );
  MessageBox ( NULL, (LPSTR)buff, (LPSTR)"Mouse", MB_OK);
}
```

CodeView für Windows:

Ein interaktives Debugging-System für Windows

Microsoft Windows und CodeView sind zwei Produkte, die wegen ihrer stark interaktiven Benutzeroberfläche sichtbare Auswirkungen auf die PC-Welt hatten. Das Benutzerinterface von CodeView basiert auf zeichenorientierten Fenstern, und bietet einen Bedienungskomfort, der bisher bei einem Debugger für Microcomputer nicht erhältlich war. Das Windows-Benutzerinterface ist ein vorgeschlagener Bedienungsstil und ein Paket von mächtigen Hilfsmitteln, die für die seine Implementierung benötigt werden.

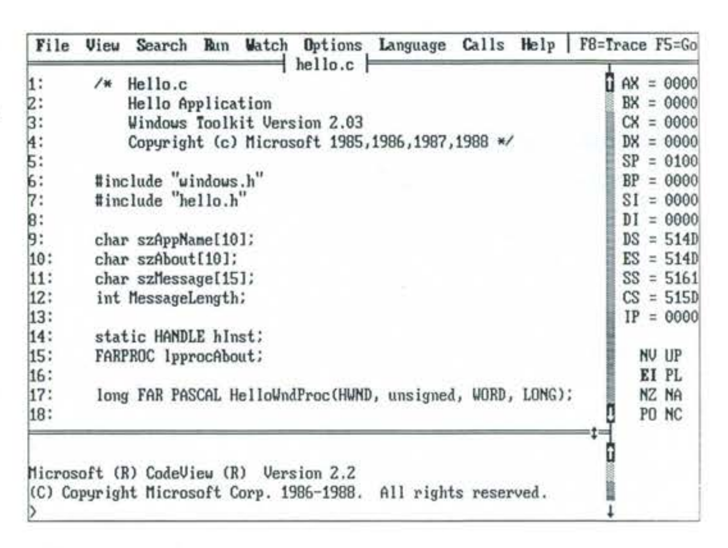


Bild 1

Dieser Artikel beschreibt CodeView für Windows (CVW), ein Produkt, das vielen DOS-Benutzern bekannt ist, das bis jetzt aber nicht mit Windows lief. Zuerst wollen wir sehen, warum ein interaktiver Debugger für den Programmierer hilfreich ist. Nach einer genauen Beschreibung der Hardwarevoraussetzungen für CVW werden wir uns eine Beta-Version von CVW ansehen. Zu guter Letzt zeigt eine Beispielsitzung, wie die an das Hauptfenster einer Applikation gelangenden Nachrichten überwacht werden können.

CodeView bietet viele der von Windows-Applikationen bekannten Neuerungen (*Bild 1*), wie Drop-Down Menüs, Dialogboxen, Fenster und die Unterstützung der Maus. Es ist paradox, daß Windows-Programmierer dieses Hilfsmittel so lange nicht benutzen konnten.

Es gibt aber eine einfache Erklärung hierfür. Die beiden Produkte wurden unabhängig voneinander von zwei verschiedenen Entwicklungsgruppen bei Microsoft entwickelt. CodeView wurde ursprünglich nicht für die Verwendung unter Windows entwickelt. Er diente der Entwicklung von DOS-Programmen.

Obwohl das Interface dem von Windows gleicht, ist CVW keine Windows-Applikation im gewöhnlichen Sinne. Statt dessen ist es eine Spezialversion des Original CodeView, dem beigebracht wurde, wie die Speicherverwaltung von Windows arbeitet. Was Ihnen zuerst auffallen wird: CVW teilt sich den Bildschirm nicht mit Windows. CVW sendet seine Ausgaben an seinen eigenen Bildschirmmonitor, in der gleichen Weise wie Symdeb, ein anderer interaktiver Debugger.

Interaktive Debugger

In den Windows-Programmierklassen, die wir unterrichten, sind wir immer wieder erstaunt über die Anzahl der Leute, die noch nie einen interaktiven Debugger benutzt haben. Es gibt sicher Alternativen zu einem interaktiven Debugger, so wie das Einfügen von Aufrufen von `MessageBox` zur Anzeige der Daten und der Aufruf von `fprintf`, um Traceinformationen an den Kommunikationsport auszugeben. Diese Methoden sind aber kümmerlich, wenn man Sie mit den Möglichkeiten und der Flexibilität eines interaktiven Debuggers vergleicht, wobei Sie die volle Kontrolle über den Computer haben, während Ihr Programm läuft. Ohne interaktiven Debugger müssen Sie Ihr Programm stoppen, umprogrammieren und wieder kompilieren, bevor Sie einen neuen Versuch wagen können. Ein interaktiver Debugger bietet Ihnen die Freiheit, zu entscheiden, wie Sie Ihr Programm erforschen wollen. Er erlaubt augenblickliche Eingriffe in ein laufendes Programm auf einer Ebene, die sonst nicht möglich ist.

Das anfängliche Arbeiten mit einem interaktiven Debugger ist aber nicht ganz einfach. Der Aufbau der Debuggerumgebung erfordert Zeit, und Sie müssen ein wenig mit dem Debugger vertraut sein, sonst ist er nicht wirklich hilfreich. Durch die strengen Entwicklungsvorgaben, die Programmierer haben, wird häufig keine Testumgebung geschaffen. Es wäre aber dennoch die Zeit wert. Leute, die mit einem interaktiven Debugger arbeiten, haben ziemlich bald große Vorteile.

Entwicklungsprobleme

Sogar mit einem interakiven Debugger bietet das Debuggen einer Windows-Applikation einige Herausforderungen. Es wäre schon schlecht genug, wenn das dynamische Laden, Verschieben und Auslagern von Speicherobjekten die einzige Quelle der Probleme wäre. Doch die Struktur von Windows-Programmen, das Fehlen eines Hardware-Speicherschutzes, und die Tatsache, daß man sich mit so vielen Sachen wie den Routinen des Software Development Kits (SDK), der Intel-Architektur und der Programmierung in C gut auskennen muß, führen zu den Schwierigkeiten, die beim Programmieren und Testen einer Windows-Applikation auftreten.

Für einige dieser Probleme ist die einzige Lösung Zeit und Geduld. Das Programmieren unter Windows verlangt von Ihnen alles über das Programmieren zu wissen, und trotzdem scheint es so, als wäre alles was Sie wissen falsch.

Das Nutzen der Vorteile der Programmierung unter Windows/Presentations Manager erfordert von Ihnen das Erlernen neuer Denkweisen. Sechs Monate intensive Windows-Programmierung, unter der Voraussetzung, daß Erfahrungen in C vorhanden sind, ist ein guter Anhaltspunkt.

Während Sie diese Erfahrungen machen, sind einige Fehlerprüfungen jedoch kostenlos, sowohl vom Compiler, als auch von Windows selbst. Dies erfordert nur das Setzen der richtigen Schalter. Hier sind einige Vorschläge, um die Probleme zu überwinden, die sich beim Programmieren unter Windows ergeben.

Verwenden Sie immer die höchste Warnungsstufe beim Übersetzen (W2 oder W3) und streben Sie nach einer fehlerfreien Übersetzung.

Deklarieren Sie jede Funktion die Sie schreiben. Sehen Sie hierzu auch die Beispiele in WINDOWS.H, wo Deklarationen wie

```
BOOL FAR PASCAL TextOut(HDC, short, short, LPSTR, short);
```

vorhanden sind. Diese Deklaration gibt dem Compiler wertvolle Informationen, wie den Rückgabewert (im Beispiel BOOL), die Aufrufkonvention (FAR und PASCAL), die Anzahl der Parameter (5) und den Typ jedes Parameters.

Wenn Sie Ihre Funktionen deklarieren, dann können Sie schnell und leicht einige Programmierfehler in C vermeiden. In der Entwicklungsphase werden beispielsweise oftmals Parameter zu Funktionen hinzugefügt. Wenn Sie es aber versäumen, jeden Funktionsaufruf zu ändern, enthält Ihr Programm einen Fehler.

Dieser Fehler fällt auf Sie zurück, indem Ihr Programm sofort Fehler macht. In diesem Fall werden Sie vermutlich suchen, bis Sie den Fehler finden. Oder aber der Fehler tritt nicht sofort auf und verursacht dann zufällige und unvorhersagbare Probleme. Da diese Art von Fehler vermieden werden kann, erspart Ihnen das Deklarieren Zeit und sollte immer in Verbindung mit den Compilerschaltern W2 oder W3 benutzt werden.

Vermeiden Sie den Einsatz von Casts. Es ist in vielen Situationen überflüssig, und kann Fehler verbergen. Lassen Sie sich vom Compiler die Typunterschiede anzeigen, und verwenden Sie dann nur an diesen Stellen Casts.

Der folgende Einsatz eines Casts ist überflüssig:

```
hDC = BeginPaint(hWnd, (LPPAINTSTRUCT)&ps);
TextOut(hDC, 10, 10, (LPSTR)"Es ist ein weiter Weg", 20);
EndPaint(hWnd, (LPPAINTSTRUCT)&ps);
```

Dies war aber in den früheren Versionen von Windows notwendig, da der Compiler zu dieser Zeit noch keine Prototyp-Deklarationen verarbeitete. Jede Version des Microsoft C Compilers seit der Version 4.0 unterstützt Prototyp-Funktionsdeklarationen, und benötigt deshalb in vielen Fällen keine Casts. Die Deklaration gestattet es dem Compiler, einen Near-Zeiger in einen Far-Zeiger zu wandeln.

Vergewissern Sie sich, daß Sie jede Fenster-Prozedur, Dialogbox-Prozedur, Unterklassen-Prozedur in ein Programm einbeziehen (include), die in der EXPORTS-Liste Ihrer Moduldefinitionsdatei (DEF) enthalten ist. Dies ist ein häufiger Fehler, der viel Zeit kostet. Das Fehlen von Prüfmöglichkeiten vom Compiler und Linker in dieser Beziehung bewirkt, daß dieser Fehler schwer zu finden ist.

Installieren und benutzen Sie eine Debugversion von Windows, nicht zu verwechseln mit einem zu Windows kompatiblen Debugger, wie CodeView. Sie erreichen dies, indem Sie spezielle Versionen von KERNEL.EXE, USER.EXE und GDI.EXE (aus dem SDK) verwenden.

Die Debugversion führt Fehlerprüfungen durch, die in der normalen Version von Windows nicht verfügbar sind. Sie handhabt Probleme, wie ungültige Handles und wilde Zeiger und zeigt Ihnen dieses über ein Debugterminal an, das Sie zum Erhalten der fatalen Fehlermeldungen, auch RIP-Codes (Rest In Peace) genannt, bereithalten müssen. Vergewissern Sie sich auch, daß die Dateien KERNEL.SYS, USER.SYM und GDI.SYM vorhanden sind, die benötigt werden, um die Stack-Trace-Informationen anzuzeigen, wenn der RIP-Code erzeugt wird.

Wenn Sie die Debugversion installieren, vergewissern Sie sich auch, daß die folgenden Schalter in der Datei WIN.INI gesetzt sind:

```
[kernel]
EnableHeapChecking=1
EnableFreeChecking=1
```

Dies bewirkt das Füllen aller freien Bytes mit dem Wert CCH. Das schützt Sie auf zweifache Art. Erstens, wenn ein wilder Zeiger in Ihrem Programm in einen freien Block schreibt, wird eine fatale Beendigungsmeldung an Ihr Debugterminal gesendet. Zweitens sind Sie geschützt, wenn Ihr Programm einen freien Block anspringt. Der Wert CCH entspricht dem Maschinenbefehl INT 3, was den Aufruf des Debuggers zur Folge hat. Ein wilder Sprung aktiviert den Debugger und ermöglicht Ihnen so, den Stack anzusehen, und den Programmteil ausfindig zu machen, der zum Absturz führte. Wegen dieser Vorteile wird der Gebrauch dieser Schalter sehr empfohlen.

Wenn Sie Expanded Memory (EMS) oder Windows/386, das eingebaute EMS Unterstützung bietet, verwenden, vergewissern Sie sich, daß die folgenden Schalter in der Datei WIN.INI gesetzt sind:

```
[kernel]
EnableEMSDebug=1
```

Dieser Schalter bewirkt, daß dem Debugger mitgeteilt wird, wenn EMS-Speicherseiten gewechselt werden. Sie können so auch Breakpunkte in den auslagerbaren Codesegmenten der dynamischen Linkbibliotheken setzen, wo sich viele SDK-Routinen befinden.

```
/
/ Standard command line definitions
/
cp=cl -d -c -AS -Gsw -Od -Zpie
/
/ Standard inference rules
/
/.c.obj:$(cp) $
.c
/
/ The C File List
/

hello.obj: hello.c hello.h

hello.res: hello.rc hello.ico hello.h
rc -r hello.rc

hello.exe: hello.obj hello.res hello.def
link4 hello/CO,/align:16,,slibw/NOE,hello.def
rc hello.res
```

Bild 2

Hardwarevoraussetzungen

CVW stellt einige Anforderungen an die Hardware. So ist das Vorhandensein eines zweiten Monitors Voraussetzung. Wir werden uns die verfügbaren Möglichkeiten ansehen, und die Notwendigkeit von EMS-4.0-Speicher diskutieren. Wenn Sie SymDeb verwendet haben, ist die Installation des zweiten Monitors dieselbe, und Sie sind bereit für CVW.

Sie haben zwei Möglichkeiten. Einen monochromen Bildschirmadapter (MDA) mit einem Monochrommonitor oder ein asynchrones Terminal mit einem Nullmodem-Adapter. Die erste Lösung ist vorzuziehen, da Sie CVW dann im Ganzbildschirmmodus benutzen können. Ein asynchrones Terminal ist hilfreich, da Sie Vorteile beim Zugriff auf die lokalen Variablen haben, die CodeView unterstützt. Sie müssen aber im Zeilenmodus ohne Ganzseitenanzeige arbeiten.

Wenn Sie im Ganzseitenmodus arbeiten, können Sie mit CodeView die Maus benutzen, um Menüpunkte anzuwählen, und seine Aktionen kontrollieren (wenn CodeView die Kontrolle hat). Auf diese Art arbeitet Codeview wunderbar, indem es die Maus mit Windows teilt. Sie müssen aber den DOS-Maustreiber laden, den CVW dann für seine Mausunterstützung benötigt. Sogar wenn Sie keine Maus benutzen (zum Beispiel im Zeilenmodus), müssen Sie dennoch einen Treiber installieren. Ansonsten funktioniert die Maus in Windows nicht mehr, obwohl Sie weiterhin Windows mit der Tastatur bedienen könnten. Wir hoffen, daß dieses Problem vor der Auslieferung von CVW behoben ist.

```
C> cvw /2 /L hello.exe \windows\win.com hello.exe
C> cvw /C=com1 /T /L hello.exe \windows\win.com hello.exe
```

Bild 3

EMS-4.0-Speicher

CVW verlangt EMS-4.0-Speicher im System. CVW braucht bis zu 138 Kilobyte Hauptspeicher, und kann darüber hinaus EMS-Speicher für Code, Daten und Symboltabellen verwenden. Die verfügbaren Optionen hängen von der verwendeten Version von Windows ab. Mit Windows 2.x müssen Sie ein EMS-Board oder einen EMS-Treiber der Version 4.0 oder später verwenden. Wir unternahmen Versuche mit einem Intel Above Board und mit einem AST RAMpage Board. Beide arbeiteten fehlerfrei, aber Sie müssen auf die richtige Version des Treibers achten.

Windows/386 enthält EMS-4.0-Unterstützung, die auch von CVW verwendet werden kann. Hierbei müssen Sie aber zwei Dinge beachten. Erstens müssen Sie mindestens 640 Kilobyte Hauptspeicher und 512 Kilobyte Extended Speicher haben, damit der EMS-Speicher überhaupt unter Windows/386 verfügbar ist. Zweitens müssen Sie einen Trick anwenden, damit die Unterstützung von EMS zuerst geladen wird. Sie starten dann CVW, bevor Sie Windows starten. Dieser Vorgang ist detailliert in der Dokumentation beschrieben.

CodeView und Windows

Nun da CodeView verfügbar ist, um Windows-Applikationen zu debuggen, werden sich die Windows-Programmierer fragen, ob seine Fähigkeiten ausreichen, um die extrem komplexen Windows-Applikationen zu untersuchen. Die Antwort ist ja. Lassen Sie uns CodeViews Stärken und Schwächen in der Windows-Umgebung untersuchen, wobei zu bedenken ist, daß wir eine Beta-Version verwenden.

Windows-Programme

Jeder Windows-Programmierer lernt sehr schnell, daß eine Windows-Applikation sich nicht selbst kontrolliert. Sie lebt vielmehr von den Nachrichten, die der Benutzer, Windows oder andere Programme schicken. Windows-Programme haben nicht immer Kontrolle über den Prozessor, stattdessen sind Sie meistens unterbrochen, und warten auf ankommende Nachrichten.

Wird eine Nachricht an ein Windows-Programm übergeben, bekommt dieses den Prozessor zugeteilt, und muß auf diese Nachricht antworten. Die Nachricht wird oftmals an die `GetMessage`-Bearbeitungsschleife übergeben. Von hier geht die Kontrolle an die entsprechende Fensterfunktion. Die Entscheidung basiert auf einem Teil der Nachricht, der Fenster-Handle.

Wenn eine Windows-Funktion eine Nachricht erhält, untersucht Sie den Nachrichtentyp und springt in der Regel in eine Nachrichtenfunktion genannte Prozedur. Jede Nachrichtenfunktion ist in der Beziehung unabhängig von der anderen, da Sie nicht weiß, ob andere Nachrichten schon eingegangen sind oder folgen werden.

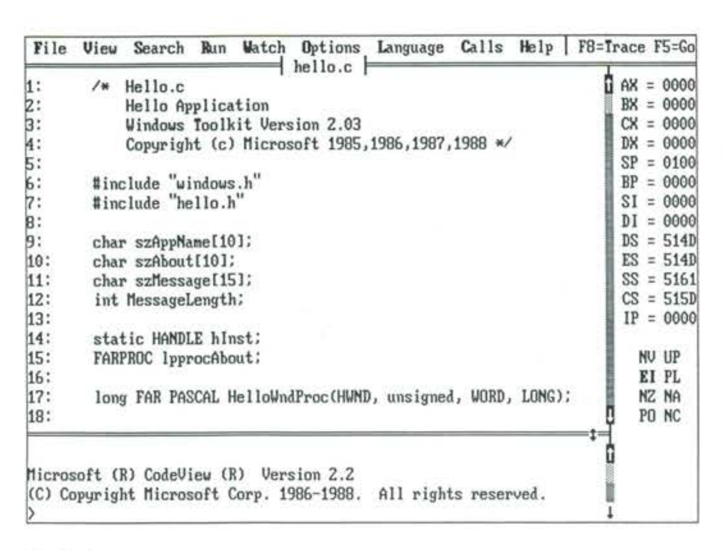


Bild 4

```
D:\HELLO>cvw /2 /L hello.exe c:\windows\win.com hello.exe
```

Bild 5

Sie kann nur den augenblicklichen Zustand der Applikation testen, zum Beispiel die aktive Zelle einer Tabellenkalkulation, und entsprechend reagieren.

Das klassische Beispiel einer Windows-Nachricht, auf die Applikationen reagieren müssen ist `WM_PAINT`. Als Antwort darauf muß eine Fenster-Funktion in jedem Fall seinen Client-Bereich neu darstellen. In dieser Beziehung reagiert ein Windows-Programm nur. Es ist der Benutzer, nicht das Programm, der den Programmablauf bestimmt.

Ein Windows-Programm kann auch als Automat betrachtet werden, dessen Zustand in externen Variablen abgespeichert ist. Windows-Programme unterscheiden sich auch von normalen Programmen dadurch, daß Sie objektorientiert sind. Fensterklassen, Fenster, Anzeigekontexte, GDI-Objekte und geladene Ressourcen sind alle Objekte. Sie werden durch eine Handle identifiziert, und Sie brauchen sich nicht um die zugehörige Datenstruktur kümmern. Ein gegebenes Objekt, wie etwa ein Fenster, kann in einer bestimmten Art verändert werden (jedes Fenster kann verschoben, in der Größe verändert, neu gezeichnet oder mit einer Nachricht versehen werden).

Windows ist auch eine bibliothekenorientierte Umgebung (speziell dynamische Linkbibliotheken). Alle Entwicklungskit-Routinen sind in dynamischen Linkbibliotheken enthalten. Sie können dynamische Linkbibliotheken verwenden, um Code, Ressourcen, Daten und die Kontrolle über die Allokierung beschränkter Ressourcen, wie Systemspeicher, oder eines Netzwerkadapters gemeinsam zu verwenden.

Das Debuggen mit CodeView

Programme bestehen aus zwei Teilen, Code und Daten. Einer der Hauptgründe für die Verwendung eines interaktiven Debuggers ist, Ihnen die Möglichkeit zu geben, die Wege durch den Code zu verfolgen. Während dieser Reise können Sie die Daten, die Ihr Code bearbeitet, sehen und sogar manipulieren. CodeView bietet drei Möglichkeiten die Ausführung zu verfolgen. Die zeilenweise Ausführung, Ausführung in reduzierter Geschwindigkeit, und die Ausführung in voller Geschwindigkeit, wobei aber die Ausführung an einer bestimmten Stelle im Code oder abhängig von vorher festgelegten Datenbedingungen angehalten werden kann. Die zeilenweise Ausführung ist entweder auf Quellcode oder auf Objektebene möglich. Die Ausführung in reduzierter Geschwindigkeit bedeutet ungefähr 4 Anweisungen je Sekunde, wobei ständig die Anzeige von Code und Daten synchron aktualisiert wird.

Ein in reduzierter Geschwindigkeit ablaufendes Programm kann ständig angehalten werden, um andere Aktionen mit dem Debugger durchzuführen. Von jetzt an werden wir die Ausführung in reduzierter Geschwindigkeit als Ausführen des Programms, das Ausführen in voller Geschwindigkeit dagegen als Laufenlassen bezeichnen.

Die dritte Technik, das Laufenlassen eines Programms mit voller Geschwindigkeit und das Anhalten an vorher bestimmten Punkten verlangt das Setzen von Break-, Watch- und Tracepunkten. Das Setzen eines Breakpunktes bedeutet die Angabe einer Stelle des Codes, an der das Programm halten soll. Dies wird durch Anklicken der entsprechenden Zeile in der Codeanzeige erreicht oder durch die Verwendung des Breakpoint-Kommandos (bp).

Die Zeile, an der der Breakpunkt steht, wird hervorgehoben angezeigt. Sie können den Breakpunkt löschen, indem Sie ihn noch einmal anklicken. Anschließend ist die Darstellung dieser Zeile wieder normal. Es können gleichzeitig maximal 19 Breakpunkte gesetzt werden. Eine Ungereimtheit gibt es bei dieser Sache. CVW zeigt Zeilen mit Breakpunkten intensiv an. Die aktuelle Zeile beim Tracen ist aber auch intensiv dargestellt. Diese Darstellung macht es schwierig, herauszufinden, ob in der aktuellen Zeile ein Breakpunkt ist, oder nicht. Um dies zu erkennen, müssen Sie an eine andere Stelle tracen.

```
File View Search Run Watch Options Language Calls Help | F8=Trace F5=Go
                              hello.c
138:
139:    /* Procedures which make up the window class. */
140:    long FAR PASCAL HelloWndProc( hWnd, message, wParam, lParam )
141:    HWND hWnd;
142:    unsigned message;
143:    WORD wParam;
144:    LONG lParam;
145:    {
146:        PAINTSTRUCT ps;
147:
148:        switch (message)
149:        {
150:        case WM_SYSCOMMAND:
151:            switch (wParam)
152:            {
153:            case IDSABOUT:
154:                DialogBox( hInst, MAKEINTRESOURCE(ABOUTBOX), hWnd, lpprocAb
155:                break;

Microsoft (R) CodeView (R)  Version 2.2
(C) Copyright Microsoft Corp. 1986-1988.  All rights reserved.
>bp .148
>
```

Bild 6

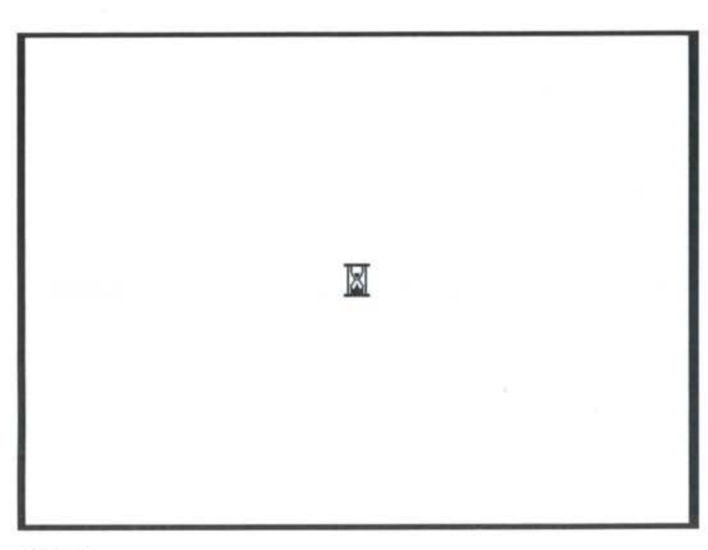


Bild 7

So wie Breakpunkte Haltepunkte sind, die im Code gesetzt werden, sind Watch- und Tracepunkte Haltepunkte, die auf Datenwerten basieren. Ein Watchpunkt ist ein bedingtes Statement, das eine oder mehrere Variable enthält, wie z.B. `count==80`. Die Programmausführung hält an der Stelle an, an der der Watchpunkt wahr wird.

Ein Tracepunkt bezieht sich auch auf eine oder mehrere Variablen, aber er muß kein bedingtes Statement sein. Das Programm hält immer an, wenn sich der Wert des Tracepunktes verändert, unabhängig von seinem Wert.

Ein schöner Effekt bei Break-, Watch- und Tracepunkten ist, daß Sie erhalten bleiben, sogar nach dem Ende des Programms. So können Sie diese setzen und das Programm mehrere Male laufen lassen. Lokale Variablen sind bei Watch- und Tracepunkten nicht ganz so schön. Die lokalen Variablen sind nur innerhalb der Funktion gültig, in der sie definiert sind, und deshalb sind sie CodeView auch nur zu dieser Zeit bekannt. Wenn Sie also einen Watch- oder Tracepunkt auf eine lokale Variable setzen wollen, müssen Sie zuerst einen Breakpunkt bei der Funktion setzen, und das Programm bis zu diesem ausführen. Jetzt können Sie den Watch- oder Tracepunkt auf die lokale Variable setzen, löschen, oder das Programm weiter ausführen. Eine Einschränkung von Codeview und jedem anderen interakiven Debugger ist, daß die Eingaben zwischen dem Programm und dem Debugger aufgeteilt werden müssen. Natürlich benötigt Ihr Programm Eingaben, die es entweder durch die Maus oder von der Tastatur erhält. Und ebenso benötigt der Debugger Eingaben. Im Falle von CodeView können sowohl Maus als auch Tastatur benutzt werden. Das Debuggen der Maus- oder Tastatureingaben kann jedoch schrecklich sein. Natürlich gibt es Auswege.

Zum Beispiel kann das Setzen eines Breakpunktes zur Überwachung der Bearbeitung der Nachricht `WM_MOUSEMOVE` sehr schwierig, wenn nicht sogar unmöglich sein. Eine andere Einschränkung bei jedem interaktiven Debugger ist das Testen von zeitkritischen Anwendungen.

Daher ist CVW bei der Verwendung mit Programmen, die die Nachricht `WM_TIMER` benötigen, eingeschränkt. Wenn Ihr Programm externe Echtzeitereignisse verfolgen muß, so ist dieser Teil der Applikation wahrscheinlich mit einem interaktiven Debugger nicht zu testen.

Das Debuggen unter Windows

Aufgrund der Tatsache, daß Windows-Programme auf Nachrichten reagieren, können sie als Automaten betrachtet werden, und außerdem befinden Sie sich in einer objektorientierten und dynamischen Linkbibliotheken-Umgebung. Wie kann uns hier CVW bei der Fehlersuche helfen? CVW behandelt die reagierende Natur eines Windows-Programms sehr gut, vor allem wegen seiner ausgezeichneten Fähigkeiten, den Programmablauf zu verfolgen. Durch das Setzen von Break-, Watch- und Tracepunkten können Sie sehen, welche Arten von Nachrichten ankommen, und was das Programm mit diesen Nachrichten anfängt. Später werden wir dies tun, indem wir den Nachrichtenverkehr untersuchen, der beim Erzeugen des Hauptfensters einer Applikation anfällt.

CVW paßt auch gut zu der Automatennatur einer Windows-Applikation. Da der Zustand in Programmvariablen festgehalten wird, und der Benutzer sieht, wie sich diese Variablen beim Programmlauf verändern. Der Benutzer kann sogar anhalten, wenn die Variablen einen unerwarteten Wert annehmen. Er kann sich auch vom Zustand des Programmes überzeugen, wenn es läuft, und so Probleme in diesem Bereich erkennen. CVW hat nur begrenzten Wert bei der Behandlung der objektorientierten Natur eines Windows-Programms. Dies ist nicht so sehr eine Schwäche von CodeView, sondern rührt daher, daß Objekte außerhalb des Datensegments des Programms gehalten werden. Stifte, Pinsel, Zeichensätze, Bitmaps, global allokierter Speicher, und die Datenstruktur, die ein Fenster beschreibt, entziehen sich dem leichten Zugriff durch CodeView.

```
File  View  Search  Run  Watch  Options  Language  Calls  Help | F8=Trace F5=Go
                              hello.c
0)  message,x  :  0024

139:    /* Procedures which make up the window class. */
140:    long FAR PASCAL HelloWndProc( hWnd, message, wParam, lParam )
141:    HWND hWnd;
142:    unsigned message;
143:    WORD wParam;
144:    LONG lParam;
145:    {
146:        PAINTSTRUCT ps;
147:
148:        switch (message)
149:        {
150:        case WM_SYSCOMMAND:
151:            switch (wParam)
152:            {
153:            case IDSABOUT:
154:                DialogBox( hInst, MAKEINTRESOURCE(ABOUTBOX), hWnd, lpprocAb
155:                break;
156:            default:

>w? message,x
>
```

Bild 8

```
File  View  Search  Run  Watch  Options  Language  Calls  Help | F8=Trace F5=Go
                              hello.c
0)  message,x  :  0081

139:    /* Procedures which make up the window class. */
140:    long FAR PASCAL HelloWndProc( hWnd, message, wParam, lParam )
141:    HWND hWnd;
142:    unsigned message;
143:    WORD wParam;
144:    LONG lParam;
145:    {
146:        PAINTSTRUCT ps;
147:
148:        switch (message)
149:        {
150:        case WM_SYSCOMMAND:
151:            switch (wParam)
152:            {
153:            case IDSABOUT:
154:                DialogBox( hInst, MAKEINTRESOURCE(ABOUTBOX), hWnd, lpprocAb
155:                break;

>w? message,x
>g
>
```

Bild 9

Wenn Sie die Inhalte eines Speicherobjektes, eines Cursors, eines Sinnbilds oder einer Bitmap sehen wollen, bietet der Heapwalker die entsprechende Unterstützung. Wenn Sie etwas über ein GDI-Objekt erfahren wollen, können Sie die Routine `GetObject` aufrufen, um die Werte in eine Programmvariable zu übertragen, die dann mit Codeview angesehen werden kann. Wenn Sie Informationen über ein anderes Objekt, wie etwa ein Fenster sehen wollen, so müssen Sie die zugehörige Toolkit-Routine, wie `GetWindowWord` benutzen, und die Informationen in eine Programmvariable übertragen. CVW ist in einer objektorientierten Umgebung nützlich, da Sie Handles und Zeiger ansehen können, die innerhalb Ihres Programms gehalten werden. Durch das Setzen von Tracepunkten können Sie Veränderungen der Werte der Handles erkennen, und Sie sind in der Lage, Ereignisse wie das unbeabsichtigte Überschreiben einer Handle oder eines Zeigers zu finden. Sie sehen auch, wenn eine NULL-Handle zurückgegeben wird, was einen Fehler bedeutet.

CVW unterstützt Sie auch beim Arbeiten mit dynamischen Linkbibliotheken. CVW gestattet Ihnen das Tracen auf Sourcecode-Ebene in Ihren dynamischen Linkbibliotheken, was Ihnen ein genaueres Bild über die Ausführung Ihrer Applikation vermittelt.

Wenn Sie den Code in einer dynamischen Linkbibliothek von Windows, wie etwa KERNEL, USER, oder GDI sehen möchten, erlaubt Ihnen CVW auch dieses. Seine Nützlichkeit hängt aber davon ab, ob Sie die Assemblersprache des 8086 verstehen, oder nicht. Da Sie den Quellcode zu den dynamischen Linkbibliotheken nicht haben, können Sie nur auf Maschinencode-Ebene tracen. Wenn Sie einigermaßen mit dem 8086-Assembler vertraut sind, kann dieses helfen, zu erkennen, was eine bestimmte Routine unter gewissen Bedingungen macht.

Eine Beispielsitzung

Wir sind jetzt bereit, uns eine Beispielsitzung mit CVW anzusehen. Unsere Absicht ist, Ihnen zu zeigen, wie das Setzen eines Breakpunktes auf die `switch`-Anweisung der Fenster-Prozedur es gestattet, die Nachrichten, die beim Erzeugen eines Fensters anfallen, zu beobachten.

Der Beginn

Das Programm, das wir untersuchen, ist das Beispielprogramm »Hello Windows« von der Beispieldiskette des SDK Version 2.03, das Sie vom Windows SDK kopieren sollten, um die Debugsitzung nachvollziehen zu können.

Der erste Schritt ist die Änderung der Make-Datei des Programms, damit Sie die benötigten Compiler- und Linker-Schalter enthält. Das bedeutet das Hinzufügen der Schalter `/Zi` und `/Od` zu den Compiler-Optionen und `/CO` zu den Linkeroptionen. Die komplette Make-Datei sehen Sie in *Bild 2*.

Nachdem Sie `HELLO.EXE` neu erzeugt haben, starten Sie CVW durch einen der beiden im *Bild 3* gezeigten Aufrufe. Wenn Sie einen MDA-Adapter mit einem monochromen Bildschirm benutzen, der Ihnen das Debuggen mit Ganzseitenanzeige erlaubt, dann sollten Sie den ersten Aufruf benutzen. Benutzen Sie aber ein asynchrones Terminal mit einem Nullmodem-Adapter als Debug-Terminal, was Ihnen nur den Zeilenmodus ermöglicht, so müssen Sie den zweiten Aufruf benutzen.

Im Rest dieser Sitzung wird der Ganzseitenmodus benutzt. Beim Start von CVW sehen Sie die beiden Bildschirmanzeigen in den *Bildern 4 und 5*. Im *Bild 4* sehen Sie, daß Codeview gestartet ist. Im *Bild 5* jedoch sehen Sie, daß Windows selbst noch nicht gestartet ist.

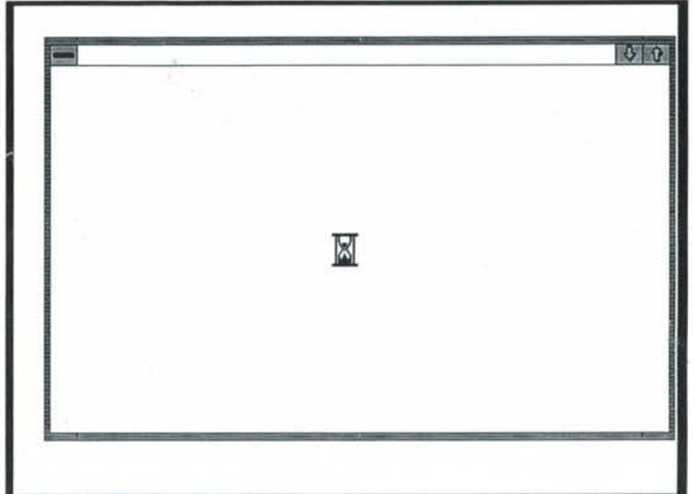




Bild 10

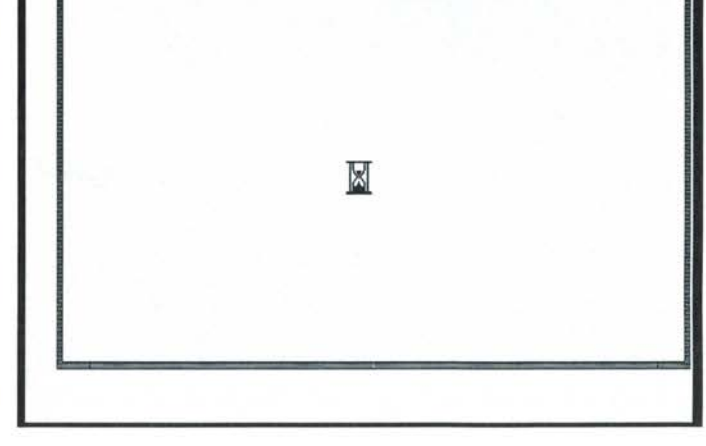


Bild 11

Setzen der Breakpunkte

Setzen Sie zuerst einen Breakpunkt am `switch`-Statement der Fenster-Prozedur, was bedeutet, daß Sie zuerst die Quellcodezeile suchen müssen und sie dann entweder mit der linken Maustaste anklicken, oder das BP-Kommando benutzen und die entsprechende Zeile angeben. *Bild 6* zeigt den gesetzten Breakpunkt mit dem intensiv dargestellten BP-Kommando.

GO

Führen Sie dann das Programm aus. Bei der Eingabe des GO-Kommandos sehen Sie, wie zuerst Windows startet. *Bild 7* zeigt die Bildschirmanzeige nachdem Windows gestartet ist, und das System den Breakpunkt bei dem `switch`-Statement erreicht hat. Beachten Sie, daß der Windows-Bildschirm das vertraute Blau anzeigt, obgleich nichts darauf abgebildet ist. Auf der Anzeige von CodeView sehen Sie, daß die Maus aktiv ist und Sie Kommandos eingeben können.

Die Watch-Liste

Jetzt werden wir die lokale Variable `message` zu der Watch-Liste hinzufügen. Da wir die Variable hexadezimal angezeigt bekommen wollen, hängen wir einen Typmodifizierer an das `w?`-Kommando an. Wir fügen `message` zu der Watch-Liste, indem wir beim CodeView-Bereitzeichen folgendes eingeben:

```
> w? message, x
```

Bild 8 zeigt, daß das Watch-Fenster geöffnet wurde, und daß der augenblickliche Wert von `message` (`0024H`) angezeigt wird. Ein kurzer Blick in die Datei `WINDOWS.H` zeigt daß dieser Wert die Nachricht `WM_GETMINMAXINFO` bedeutet. Obwohl unser Fenster eine Nachricht erhalten hat, wurde Sie bis jetzt noch nicht bearbeitet. Um die Nachricht zu bearbeiten, geben wir erneut das Kommando `g` (GO) ein, und die Ausführung stoppt am nächsten Breakpunkt. Die Bildschirmanzeige sieht jetzt wie in *Bild 9* aus. Beachten Sie, daß die CVW-Anzeige den Wert `0081H` für `message` anzeigt, und daß sich die Windows-Anzeige nicht verändert hat.

Die nächsten Nachrichten sind:



```
WM_GETMINMAXINFO
WM_NCCREATE
WM_CALCSIZE
WM_CREATE
WM_SHOWWINDOW
WM_SETVISIBLE
WM_ACTIVATEAPP
```

Der Nichtclient-Bereich

Die erste Nachricht, die etwas auf den Bildschirm bringt ist die Nachricht `WM_NCACTIVATE`. Der Teil `NC` der Nachricht steht für *Not Client*. Wie *Bild 10* zeigt, erfolgt lediglich das Zeichnen des Fensterrandes zusammen mit der Systembox und den Boxen für Minimize und Maximize.

Die nächste Nachricht, `WM_GETTEXT`, scheint die Titelzeile anzuzeigen. Tatsächlich nehmen wir an, daß das vorhergehende `WM_NCACTIVATE` eine `WM_GETTEXT`-Nachricht schickte, um diese Windows-Meldung zu erhalten, um die Titelzeileninformationen richtig anzeigen zu können.

Eine weitere Gruppe von Nachrichten, die die Anzeige beeinflußen erscheint, sind folgende:

```
WM_ACTIVATE
WM_SETFOCUS
WM_NCPAINT
WM_SYNCPAINT
```

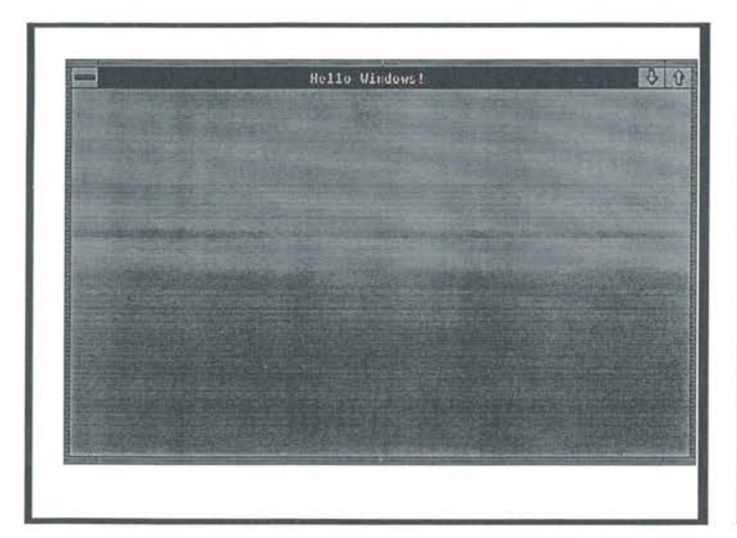


Bild 12

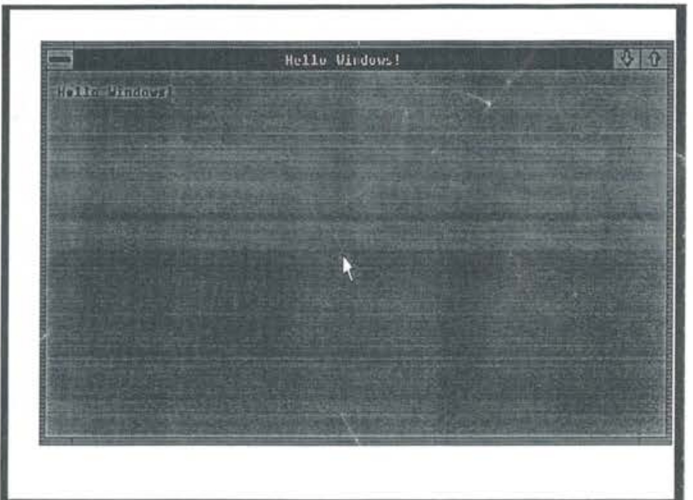


Bild 13

Löschen des Hintergrunds

Bild 12 zeigt den Effekt der Nachricht `WM_ERASEBKGND`. Der Hintergrund des Fensters ist mit der voreingestellten Hintergrundfarbe des Hauptfensters, das weiß ist, gelöscht worden. Es erscheinen weitere Nachrichten:

```
WM_SIZE
WM_MOVE
```

Sie sollten bedenken, daß diese beiden Nachrichten beim Erzeugen des Fensters ankommen, was wichtig ist, wenn Sie planen, auf diese Nachrichten antworten zu wollen. Denken Sie über diese Nachrichten, als wenn Sie sagen würden, das Fenster ändert sich in der Größe von Nichts in Etwas und bewegt sich von Nirgendwo nach Irgendwo.

Hello Windows!

Wenn zu guter Letzt die `WM_PAINT`-Nachricht ankommt, erscheint das vertraute »Hello Windows«, wie in *Bild 13.* Es scheint, als seien schon zu viele Nachrichten bei der Erzeugung eines Fensters eingetroffen, aber es kommen noch zwei weitere, nämlich:

```
WM_NCHITTES
WM_SETCURSOR
```

Die letzte ankommende Nachricht ist `WM_MOUSEMOVE`. Dies beschließt den Nachrichtenverkehr beim Erzeugen des Fensters. Zusammen sind 20 Nachrichten bei der Fensterprozedur angekommen, bevor der Prozeß der Erzeugung des Fensters abgeschlossen ist.

Jetzt wird beim Bewegen der Maus eine weitere `WM_MOUSEMOVE`-Nachricht an die Fensterprozedur der Applikation gesendet. Sie müssen den Breakpunkt löschen. Ansonsten werden Sie immer von der Mausbewegungsnachricht unterbrochen. Sie tun dies, indem Sie entweder nochmals die Zeile mit der Maus anklicken, oder durch das Kommando `BC 0`.

Vor dem Verlassen von CodeView müssen Sie Windows beenden. Dies deshalb, weil Windows das System, inklusive einiger Systemvektoren, ändert, und diese wieder hergestellt werden müssen, damit das System normal arbeiten kann. Danach können Sie CodeView mit dem QUIT-Kommando verlassen:

```
> quit
```

Ein Durchbruch

Wenn CodeView für Windows verfügbar sein wird, wird dies ein Durchbruch für Windows-Entwickler sein. Das Interface ist leicht zu handhaben. Seine Fähigkeiten, auf lokale Variablen zuzugreifen, Datenstrukturen anzuzeigen und komplexe C-Ausdrücke auszuwerten, geben ihm die Fähigkeiten einer neuen Generation von Debuggern.

Sein Speicherbedarf (138 Kilobytes) und die Notwendigkeit für EMS-Speicher kann seine Nützlichkeit für einige Entwickler jedoch einschränken. Im Gegensatz dazu benötigt SymDeb 43 Kilobytes, bietet aber nicht das Ganzseiten-Benutzerinterface und den Zugriff auf lokale Variablen, die Teil von CodeView sind, was für gewisse Windows-Applikationen immer hilfreich ist. Ebenso bietet die von uns untersuchte Kopie von CodeView nicht alle Zugriffe zu den Internas von Windows, die SymDeb bietet, wie die Ausgabe des globalen Heaps, die Anzeige der freien Liste usw. Das ist aber ein kleiner Nachteil, verglichen mit den großen Vorteilen, die das Benutzerinterface von CodeView bietet.

Paul Yao/David Durant

Die Version 5.1 des Microsoft Makro-Assemblers:

Verbesserte Makros und Sprachenschnittstellen

Interrupt Routinen, sehr schnelle asynchrone Schnittstellenroutinen, direkte Bildschirmspeicherzugriffe, direkte Diskroutinen, die Sprache C, was paßt hier nicht zusammen?

Bis jetzt werden Sie möglicherweise die Sprache C vermutet haben. Ich hätte es bestimmt. Wann immer es nötig war, eine Routine zu programmieren, die Außergewöhnliches schnell und effizient tun sollte, kam mir die Verwendung einer Hochsprache wie C nie in den Sinn. Der gesamte Overhead des Pushens des BP-Registers, sogar wenn keine Stackreferenz notwendig ist, das Verändern von Volatil-Variablen von einer es gut meinenden Optimierungsroutine und die langen Übersetzungszeiten legten mir die Verwendung des Microsoft Makro-Assemblers (MASM) nahe.

Dies bedeutete natürlich das umständliche Erzeugen der Prozedureinleitungs- und Prozedurschlußanweisungen für jede MASM-Routine. Ganz zu schweigen von der zusätzlichen Arbeit, wenn mein Programm 64 Kilobyte überschreitet, und ich auf das nächste Speichermodell umstellen muß. Ich nahm es auf mich. Ich glaubte immer, daß es zu viel Arbeit ist, ein Speichermodell-unabhängiges Programm mit MASM zu schreiben, bis ich begann, Makros zu verwenden.

Makros sind möglicherweise die am wenigsten benutzten Hilfsmittel des Assemblerprogrammierers. Ein Makro ist die Abkürzung für den öfteren Gebrauch desselben Programmteils. PUSH_ALL und POP_ALL sind sicher Dinge, die jeder Programmierer benötigt, weshalb also nicht eine Abkürzung einführen, die dann später zu den zugehörigen Anweisungen expandiert wird? Mit der Einführung von MASM 5.0 wurde das Programmieren von Makros durch einige gut geschriebene Makros erleichtert, die im Paket enthalten waren. Meine Produktivität verbesserte sich erheblich, meine Programme wurden leichter lesbar, ich brauchte nicht mehr Stunden, um auf ein anderes Speichermodell umzustellen, und ich hatte genügend Zeit, Ausreden dafür zu suchen, daß ich meine Arbeit wenig dokumentiere.

Mit der Version 5.1 wurden viele Makros überflüssig, da ihr Funktionalität jetzt im Assembler selbst enthalten ist. MASM 5.1 enthält die Änderungen, die ich auch vorgenommen hätte, wenn er von mir stammen würde.

Daher ist eventuell einiges Neucodieren Ihrer Programme notwendig. Dieser Artikel untersucht die Änderungen, die notwendig sind, um die Vorteile und Neuerungen des MASM 5.1 auszunutzen.

Die Beschränkung von C

Der Overhead in einigen C-Routinen kann ein Problem sein, wenn Sie versuchen, guten, sauberen und schnellen Code zu schreiben. Denken Sie an die Funktion `movmem`. Stellen Sie sich überlappende Quell- und Zielbereiche vor. Sie können keine so schnellen und sauberen Funktionen schreiben, die zwei Zwecke erfüllen, als wenn sie nur eine Aufgabe erledigen.

Sogar der direkte Aufruf von DOS-Interrupts ist nicht effizient. Obwohl Sie genau wissen, daß nur ein oder zwei Register vor dem Aufruf vorbesetzt sein müssen, weiß dies die `Int86`-Funktion nicht. Sie rettet alle Register, lädt diese von der `REGS`-Struktur, ruft den Interrupt auf, speichert alle Register in die `REGS`-Struktur (oder eine andere), und poppt wieder alle Register. Denken Sie an diesen Aufwand, wenn Sie nur einfach das AH-Register setzen müssen, und sogar der Rückgabewert uninteressant ist. Neben dem künstlichen Vergrößern des Programms benötigen diese überflüssigen Befehle Prozessorzeit. Das ergibt wieder den Overhead, den Sie eigentlich durch den direkten DOS-Aufruf vermeiden wollten.

Einige spezielle Operationen verlangen auch nach der Effizienz von MASM-Routinen. Speicherübertragungen zwischen verschiedenen Segmenten, das Umleiten eines Interrupts (wie des Timers, der Tastatur oder der seriellen Schnittstelle) oder etwas ausgefallenes, wie etwa ein Overlayloader.

Die Schnittstelle von MASM und C

Es gibt einige Probleme, wenn Sie aus einer MASM-Routine das Beste herausholen wollen. Verwenden Sie eine C-Funktion, die von einer Interrupt-Routine in Assembler aufgerufen wird, verursacht dies einige Verwirrung. Sie müssen eigenen Stack bereitstellen, irgendwoher das Datensegmentregister wiederherstellen und sich bewußt sein, daß der Aufruf auch nur einer C-Funktion jedes Register ändern kann.

Das Lesen der Variablen vom Stack, wenn eine C-Funktion eine Assemblerroutine aufruft, ist auch ein gutes Beispiel für experimentelles Programmieren. Ich kann mir nie den Offset im Stack merken (der vom Speichermodell und den vorher geretteten Registern abhängt), und meistens endete es mit Ungeheuern, die funktionierten, mich aber etwas entäuschten, wenn ich den Code anderen zeigte (*Bild 1*). Mit dem neuen MASM 5.1 kann dasselbe Programm einfach so geschrieben werden:

```
routine proc uses ax, var1:int
        mov     ax, var1

        <weiteres Programm>

        ret
routine endp
```

Dies ist verständlicher, obwohl der erzeugte Code nahezu identisch ist.

Ein weiterer Grund zum Ärgern ist das Poppen der Register in einer anderen Reihenfolge als das Pushen. Dies führt zu Fehlern, die man relativ schnell findet. MASM löst

dies bei Verwendung der Anweisung `uses`. Der erzeugte Code ist nicht unbedingt effizienter, aber ein solches Problem muß nicht länger durch die Verwendung des Debuggers ausgetestet werden.

MASM 5.1

Die neuen Möglichkeiten von MASM 5.1 verbessern sowohl die Formalisierung der Makros, als auch die Makrofähigkeiten selbst. Bei einem Wechsel von 5.0 auf 5.1 brauchen Sie keine großen Änderungen in Ihren Quelldateien vorzunehmen. Wenn Sie von Version 4.0 wechseln, werden Sie über die Möglichkeiten und die leichte Anwendung überrascht sein.

Makros

Makros erlauben Ihnen die Definition neuer Operanden im MASM. Statt neue zu erzeugen, ordnen Sie einem Block von Befehlen einen symbolischen Namen zu, der auch mit Variablen als Argumenten versehen sein darf. Trifft der Assembler auf ein Makro, expandiert er es zu der vorher definierten Befehlssequenz. Makros können auch etwas intelligenter sein. Sie können unterschiedliche Befehlssequenzen erzeugen, abhängig vom Argumenttyp oder der Anzahl der Argumente oder dem Speichermodell. Der Makroparser überprüft aber nur die Bedingungen zur Übersetzungszeit. Sie können nicht den Wert eines Registers zur Laufzeit verwenden. Wenn Sie ihre eigenen Makros programmieren, werden Sie diese sinnvollerweise in einer Include-Datei halten.

MASM 5.1 erlaubt die Verwendung von Textmakros, wie etwa die Verwendung des Wertes rechts einer `EQU`-Anweisung als Makro. *Bild 2* hätte in Version 5.0 einen Aufruf eines regulären Makros verlangt. Eine Möglichkeit wäre:

```
msg_print macro
      do_print %errprint
endm
do_print macro msg_list
      irp msg, <msg_list>
      push   ax
      push   dx
      mov    ah,9
      mov    dx, offset msg
      int    21h
      pop    dx
      pop    ax
      endm
endm
```

In der Version 5.1 wird jedoch das mittlere Makro nicht mehr benötigt, und ein Makro wie das folgende ist erlaubt:

```
msg_print macro
%       irp msg, <msg_list>
        push   ax
        push   dx
        mov    ah,9
        mov    dx, offset msg
        int    21h
        pop    dx
        pop    ax
        endm
endm
```

Das Prozentzeichen (%) am Anfang der `IRP`-Zeile veranlaßt die Textmakroauswertung. Es ergibt keinen verkürzten Code, aber die Anweisungen selbst sind einfacher zu verstehen.

Eine weitere nette Möglichkeit ist die Verwendung des &-Zeichens für die Stringverknüpfung außerhalb eines Makros, ebenso wie Sie es früher innerhalb eines Makros taten:

```
label equ <test_stuff>
%table_entry_&label db tester
```

In MASM 5.0 war dies nicht erlaubt, und die Zeile mit `tabel_entry` ohne & wurde vom Assembler als `table_entry_label` interpretiert und nicht ausgewertet.

Sie können jetzt auch Teilstringsuche und Größe der Textmakroargumente ermitteln. Die Verwendung des `SUBSTR`-Operators veranlaßt im *Bild 3* das Pushen der entsprechenden Register. Die folgenden Zeilen pushen die Register ax und bx:

```
reg_push 1
reg_push 2
```

Das zweite Argument ist die Startposition im String, die mit 1 beginnt, und das dritte Argument ist die Länge des Strings.

`CATSTR` erlaubt das Zusammenführen einer Reihe von Strings in einen:

```
big_string CATSTR <string1>,<string2>
```

ergibt für `big_string` `"string1string2"`.

`SIZESTR` gibt die Stringlänge zurück und ergibt für folgenden Ausdruck 14:

```
len_big_string SIZESTR big_string
```

Mit der `INSTR`-Anweisung erhalten Sie den Offset eines Substrings innerhalb eines Strings. Um zum Beispiel zu unterscheiden, ob ein Argument sowohl einen Offset als auch ein Segment enthält, können Sie nach dem Doppelpunkt an der dritten Stelle suchen:

```
print_msg macro       addr colon INSTR <addr>,<:>

if colon gt 0
msg_seg SUBSTR <addr>,1,2
len SIZESTR <addr>
lab SUBSTR <addr>,4,len - 3

        push    ax
        push    dx
        push    ds
        mov     dx, msg_seg
        mov     ds, dx
        mov     dx, offset lab
        mov     ah, 9h
        int     21h
        pop     ds
        pop     dx
        pop     ax
else
        push    ax
        push    dx
        mov     dx, offset addr
        mov     ah, 9h
        int     21h
        pop     dx
        pop     ax
endif
endm

start:
print_msg       cs:msg1
print_msg       msg1
        int     20h
        ret
msg1    db      msg1$
end start
```

Vordefinierte Makros

MASM 5.1 enthält einige vom Assembler selbst vordefinierte Makros. `@WordSize` gibt die Größe der Segmentworte in Bytes zurück. Sie erhalten 4 bei Wortgrößen von 32 Bit und 2 bei Wortgrößen von 16 Bit. Dies ist sehr hilfreich, wenn Sie die Direktive `.386` aktiviert haben und ein 386-spezifisches Makro benötigen.

`@CPU` gibt ein eindeutiges Bitmuster zurück, das den verwendeten Prozessortyp, den Modus in dem sich der 286 oder der 386 befindet (protected oder real) und die Art des installierten arithmetischen Coprozessors (falls überhaupt vorhanden) repräsentiert. Bedenken Sie aber, daß dies ein Makro ist, das zur Übersetzungszeit bewertet wird, und der Computer, auf dem das Programm später läuft, eine andere Konfiguration haben kann.

Das Makro `@Version` gibt einen drei Zeichen langen Bezeichner zurück, der angibt, welche Version von MASM Sie verwenden. Die neueste Version meldet 510. Dies ist in früheren Versionen undefiniert, die Sie mit den folgenden Anweisungen testen können.

```
IFNDEF @Version

ELSE
        Erweiterungen vorhanden
```

Zu guter Letzt sind keine `IF`-Ketten mehr als Ersatz für ein `CASE`-Statement notwendig! Sie können jetzt mit `ELSE-IF`-Anweisungen wie in *Bild 4* arbeiten.

Einige Erweiterungen für bedingtes Assemblieren erleichtern Ihnen ebenso die Arbeit. Stellen Sie sich einen Programmteil wie den folgenden vor:

```
zb proc uses bx, es, p1:ptr
; Löschen des ersten Bytes, unabhängig vom Speichermodell
if @datasize
        les     bx, p1
        ; im großen Speichermodell
else
        mov     bx, p1
        ; im kleinen Speichermodell
endif
        mov     byte ptr [bx], 0
        ret
zb endp
```

Lokale Labels

Ich hasse das Ausdenken von Variablennamen. Ich bin ein Befürworter des selbst dokumentierenden Programmierens. Variablen und Labelnamen müssen sinnbezogen sein, um Sinn beim Lesen des Programmes zu ergeben. Dies hilft während der Dokumentationsphase des Projektes. Aber das Ausdenken eindeutiger Namen ist schwierig, und der ständige Versuch, positionsunabhängige Programme zu erzeugen ist mühsam. Mit dem MASM 5.1 können Sie jetzt den Operator `@@` zur Definition von Vorwärts- oder Rückwärtssprüngen benutzen. Diese Sprünge werden zur Übersetzungszeit bewertet (*Bild 5*).

Die TYPE-Anweisung

Abhängig vom Typ einer Variablen können Sie die Operation auswählen, die auf dieses Argument oder diese Variable angewendet wird. Die `TYPE`-Anweisung wurde erweitert, um zu erkennen, ob ein Objekt programmabhängig (Code), datenabhängig, eine Konstante, ein Adressierungsmodus, ein Register (oder nicht) ist, definiert ist (oder nicht) und ob ein Symbol `public` oder lokal ist.

Die kompletten Definitionen der Bitpositionen der `TYPE`-Anweisung sind im MASM-Handbuch aufgelistet. Als Beispiel, wie nützlich die `TYPE`-Anweisung sein kann, sehen Sie hier ein Beispiel:

```
print_me macro arg
if ((.TYPE arg) AND REGISTER)
        push    arg
        push    dx
        mov     dx, arg
        call    do_print
        pop     dx
        pop     arg
elseif
        push    dx
        mov     dx, offset arg
        call    do_print
        pop     dx
endm
```

CodeView-Verbesserungen

Wenn Sie zum Debuggen CodeView benutzen, dann nützt es, wenn der Objekttyp CodeView bekannt ist. Sie können dies veranlassen, wenn Sie die verbesserten und erweiterten PTR-Datendefinitionswörter (*Bild 6*) verwenden.

Die Unterstützung von Hochsprachen

Erstens und eigentlich bin ich ein C-Programmierer. Manchmal entdecke ich, daß ich C-Syntax verwende, wenn ich in Assembler programmiere, und zögere, mich in die Niederungen des Assemblers hinunterzubegeben. Ständig denke ich an die Mühen, die mich erwarten. Das richtige Setzen der Segmentregister, das Erinnern, wo und was das Datensegment ist, das Programmieren in gemischten Speichermodellen, das Lesen der Variablen vom Stack, die Probleme mit dem Unterstrich, es ist die Mühe nicht wert.

Mit MASM 5.1 sind alle meine Ängste überflüssig. Es ist jetzt erheblich einfacher, Assembler- mit C-Programmen zu kombinieren. Obwohl mich eigentlich C interessiert, ist es doch wichtig, daß das Interface mit anderen Hochsprachen, auch mit BASIC sich erheblich vereinfacht hat.

Wie schon früher erwähnt, gestattet MASM 5.1 die Verwendung des Schlüsselworts uses als Teil der PRO-Anweisung, um den Assembler zu informieren, welche Register Ihre Routine benötigt. Beim Programm

```
my_proc proc uses ax dx
      mov    dx, offset cs:my_msg
      mov    ah, 9h
      int    21h
      ret
my_proc endp
```

wird automatisch die richtige Kombination der Befehle push ax, pop ax und push dx, pop dx für Sie erzeugt. Die Grundsyntax ist das Trennen jedes Registers durch ein Leerzeichen, einen Tabulator oder ähnliches.

Als ich meine Programme auf die Version 5.1 umstellte, verursachte dies einige Probleme. Ich verwendete eine Routine, die die Registerwerte rettete, und die Register AX und BX vertauschte. Wenn Sie irgendwelche Spielchen mit den Registerwerten treiben, die Sie vom Stack holen, müssen Sie die konventionelle Methode benützen. Oder Sie suchen die Ursache der Fehler und dokumentieren sie, oder ändern sie in einen vernünftigeren Zustand.

Etwas überraschte mich und ist augenscheinlich nirgends dokumentiert. Sie müssen die Anweisung .MODEL mit beiden Argumenten benützen, ansonsten wird die uses-Anweisung nicht richtig ausgeführt. Das erste Argument der .MODEL-Anweisung ist das Speichermodell, das Sie verwenden möchten, das zweite ist die Programmiersprache, für die Sie das Modul verwenden möchten. Sie haben die Auswahl zwischen den Speichermodellen small, medium, compact, large oder huge. Die Angabe der Sprache wie in

```
.MODEL small, C
```

generiert die Aufruf- und Rückgabesequenz für Sie, macht alle verwendeten Symbole public, und hängt an den Anfang den benötigten Unterstrich an, so daß der Linker nicht verwirrt wird. Sie müssen BASIC, C, Fortran oder Pascal angeben, da jede Sprache ihre eigenen Konventionen bezüglich Parametern und Namensgebung besitzt.

Der Update von MASM kann beim Entwickeln von übertragbarem Code hilfreich sein. Sie brauchen nur die Sprachbezeichnung in der .MODEL-Anweisung von beispielsweise C nach BASIC zu ändern und MASM ändert automatisch die Aufruf- und Parameter-Konventionen ohne zusätzliche Arbeit. Das ist für Leute, die kommerzielle Bibliotheken entwickeln, sehr hilfreich. Ein Bibliothek für kompiliertes BASIC ist jetzt genauso einfach wie für C zu entwickeln. Sie brauchen nur den Bezeichner zu ändern und neu zu übersetzen. Unterstriche erscheinen und verschwinden, wie von Geisterhand, genau so, wie es der entsprechende Sprachübersetzer zum Linken benötigt.

Stackadressierung

Wie schon angesprochen können Sie die uses-Anweisung verwenden, um anzugeben, welche Register beim Prozedureintritt gerettet und hinterher wieder vom Stack geholt werden sollen. Kombinieren Sie aber mit einer Hochsprache, wollen Sie sicherlich auch der Assemblerroutine Parameter übergeben. In den meisten Sprachen, und in ganz besonderem Maße bei C, erfolgt diese Parameterübergabe auf dem Stack. Jedes Argument der Funktion wird auf dem Stack abgelegt, entweder von links nach rechts, oder von rechts nach links (abhängig, ob sie die Funktion als Pascal deklariert haben). Die Neuerungen von MASM erlauben Ihnen den leichten Zugriff auf diese Parameter, unabhängig vom verwendeten Speichermodell. Dies erspart Ihnen Stunden der Fehlersuche und des Neuschreibens des Quellcodes, wenn Sie öfters das Speichermodell wechseln.

Nehmen wir einmal an, daß Sie nicht den Pascal-Aufruf verwenden, der die Parameter von links nach rechts auf dem Stack ablegt. Haben Sie die C-Sprache mit der .MODEL-Anweisung aktiviert, dann können Sie eine Zeile wie in *Bild 7* verwenden, die nicht nur angibt, daß das AX-Register beim Programmeintritt gerettet werden soll, sondern Ihnen auch das direkte Ansprechen von drei Parametern auf dem Stack gestattet, wie bei:

```
mov    ax, arg1
```

Dies kann unabhängig vom Speichermodell gemacht werden, und ohne den Funktionseintritts- und Endecode zu programmieren (Retten und Wiederherstellen des BP Regi-

sters), das automatisch für Sie erledigt wird.

Sie können für Ihre Argumentlisten folgende Typen auswählen: `word`, `dword`, `fword`, `qword`, `tbyte` oder sogar den Namen einer Struktur (`struc`)

Voreingestellt ist das Wort (für 8086/88/286/386) oder das Doppelwort, wenn Sie die `.386`-Anweisung verwendet haben.

Sie können den Typ ebenso mit den Zusätzen `far`, `near` oder `ptr` versehen, wie hier:

```
my_proc uses ax, arg1:far ptr dword
```

Es wird eine 4 Byte lange Variable auf dem Stack übergeben, die als `far`-Zeiger verwendet wird. Dies gestattet Ihnen die spätere Verwendung von:

```
lds    dx, arg1
```

Der Zusatz `ptr` wird aber tatsächlich vom MASM für fast nichts benützt, außer, daß zusätzliche Informationen für die Verwendung von CodeView erzeugt werden.

Die Routine in *Bild 8* zum Beispiel ergibt die Debuggeranzeige in *Bild 9* (zu der ich Kommentare angefügt habe). Derselbe Code, den Sie erzeugt haben. Augenscheinlich verwendete ich das Speichermodell `small` für dieses Beispiel.

Lokale Variablen

In einer Hochsprache wie C deklarieren Sie die lokalen Variablen am Funktionsanfang oder innerhalb eines Blocks, und der Speicherplatz wird Ihnen automatisch bereitgestellt. Am Funktionsende erfolgt die automatische Freigabe des Speichers. Sie benützen nur den Variablennamen, und der Compiler erledigt den Rest. MASM hatte schon immer diese Fähigkeiten. Ein Makro wie das folgende

```
tmp_buf equ <byte ptr [bp - size1]>
```

erlaubt Ihnen die Verwendung des Namens `tmp_buf`, als wäre es eine richtige Variable. Solche Variablen haben allerdings globale Gültigkeit, und ich verwendete stets Namen wie `tmp_buf_serial_routine` um sie von anderen `tmp_bufs` zu unterscheiden. Es wurde aber kein Platz auf dem Stack gespart, und eine unachtsame Push-/Pop-Sequenz konnte den Stack in einer sonderbaren Weise verändern. Mit MASM 5.1 können Sie jetzt Variablen mit einem nur lokalen Gültigkeitsbereich allokieren. Die Namen existieren nur innerhalb der Prozedur, in der sie definiert sind. Sehen Sie zum Beispiel *Bild 10*, wo MASM veranlaßt wird, hinter den Kulissen, fast geisterhaft, ein Textmakro zu expandieren. Hier ist ein einfaches Beispiel:

```
HEADER_SIZE    equ    3
DATA_SIZE      equ    128
CRC_SIZE       equ    2
```

Der Quellcode, der von MASM 5.1 erzeugt wird, ist in *Bild 11* zu sehen (der Cast von `tmp_buf` wird benötigt, weil `tmp_buf` als Bytearray definiert ist). Der Dump, mit den Aufrufen der anderen Routinen, der zu diesem Zweck angefertigt wurde, ist im *Bild 12* zu sehen.

Übersetzungshilfen

Die Übersetzungshilfen für Ihren nicht mit Makros versehenen MASM-Code sind sehr einfach:

Zuerst müssen Sie Ihre Push/Pop Befehlszeilen in die entsprechenden `uses`-Statements ändern. MASM ist in einer Beziehung nicht so zuvorkommend. Jeder `Ret`-Befehl erzeugt die erforderliche Pop-Befehlssequenz, deshalb sollten Sie die Routinen so ändern, daß sie nur ein `Ret`-Statement enthalten, mit Sprunganweisungen von anderen Stellen direkt auf dieses Statement. Dies ist auf jeden Fall auch gute Programmierpraxis.

Als nächstes sollten Sie alle Argumentübergaben in die neue `proc`-Anweisung ändern. Denken Sie auch daran, die Anweisungen `PUSH BP`/`MOV BP,SP`/../`POP BP` zu entfernen. MASM erzeugt diese Befehle automatisch für Sie.

Schauen Sie sich die Stellen an, an denen Sie lokalen Speicher vom Stack allokiert haben (versuchen Sie eine globale Suche nach `SUB SP`, und ändern dies in die neue lokale Aufrufkonvention.

Nun die schwierigste Aufgabe. Das Konvertieren aller Programmteile in die neuen Textmakros, um die Vorteile der Neuerungen zu nutzen. Sie müssen entscheiden, ob es sich auch lohnt, in die neue Form umzuändern.

Denken Sie daran, die *.MODEL*-Anweisung einzusetzen, sonst verhält sich MASM 5.1 wie seine Vorgänger, und Sie erhalten eine Menge Fehlermeldungen.

MASM 5.1 ist nur ein Upgrade. Ein evolutionärer, kein revolutionärer. Ein Reihe weiterer Neuerungen enthält das Paket noch, nicht nur den Assembler. Dazu gehören der neue CodeView, die Fähigkeit, leicht mit den dynamischen OS/2-Bibliotheken zu binden, die Updates zu BIND, IMPLIB, MAKE, ILINK und das Dienstprogramm EXEHDR dazu. Der Microsoft Editor gehört jetzt ebenso zu dem Paket.

Ich konnte die Anzahl meiner Quellcodezeilen um ungefähr 30% reduzieren, und ich zögere weniger, wenn ich MASM-Routinen in einem C-Programm benutzen soll. Auch brauche ich viel weniger Zeit, meine Assemblermodule zu debuggen. Jetzt kann ich die gewonnene Zeit zum Dokumentieren meiner Funktionen benutzen, sobald ich eine Routine ausgetestet habe.

Ross M. Greenberg

```
routine proc  <Name>
   push bp
   mov bp, sp
   push ax
   mov ax, ss:[bp + SPEICHER_MODEL + OFS_VAR1]
     <Anderer Code>
   pop ax
   pop bp
   ret
routine  endp
```

Bild 1: *Beispiel der alten Methode, die das Erzeugen eines User Stacks und das Pushen und Poppen der Register voraussetzte.*

```
errprint equ <err_msg1, err_msg2, err_msg3>
```

Bild 2: *Beispiel für ein Textmakro.*

```
reg_push macro    start
reg substr <axbxcxsdxsidib>,(start - 1) * 2 + 1, 2
   push reg
endm
```

Bild 3: *Beispiel für die Verwendung von* SUBSTR.

```
ELSEIF <test>                 - wenn <test> wahr ist
ELSEIF <test>                 - wenn <test> falsch
ELSEIF1                       - wenn pass1
ELSEIF2                       - wenn pass2
ELSEIFB <arg>                 - wenn Leerzeichen
ELSEIFNB <arg>                - wenn kein Leerzeichen
ELSEIFDEF <arg>               - wenn definiert
ELSEIFNDEF <arg>              - wenn nicht definiert
ELSEIFDIF <arg1>, <arg2>      - wenn unterschiedlich
ELSEIFDIFI <arg1>, <arg2>     - wenn unterschiedlich, unabhängig
                                von Groß-/Kleinschreibung
ELSEIFIDN <arg1>, <arg2>      - wenn identisch
ELSEIFIDNI <arg1>, <arg2>     - wenn identisch, unabhängig von
                                Groß-/Kleinschreibung
```

Bild 4: *Neue* ELSEIF-*Optionen.*

Impressum

Das *Microsoft System Journal* erscheint alle zwei Monate (ungerade Monatszahlen) etwa Mitte des Vormonats.
Herausgeber, verantwortlich und Anschrift der Redaktion:
Microsoft GmbH, Redaktion *Microsoft System Journal*,
Erdinger Landstr. 2, D-8011 Aschheim-Dornach
Telefon: 089 / 46107-0, Teletex/Telex: (17) 89 83 28, Telefax: 90 63 55
Redaktion:
Günter Jürgensmeier, Haar und Hartmut Niemeier, Wildenberg
Mitarbeiter dieser Ausgabe:
Marcellus Buchheit, Michael Bülow, David Durant, Ross M. Greenberg, Don Hasson, Thom Hogan, Günter Jürgensmeier, Hartmut Niemeier, Craig Stinson, Kevin P. Welch, Paul Yao
Manuskripteinsendungen:
Manuskripte und Programmlistings werden von der Redaktion gerne angenommen. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck und zur Vervielfältigung der Programmlistings auf Datenträgern. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen. Nicht zur Veröffentlichung gelangte Manuskripte und Listings können nur zurückgeschickt werden, wenn Rückporto beiliegt.
Titelgestaltung: Hermann Menig
Anzeigenverkauf: Marianne Nuß
Druck und Abonnements:
schury praxisformulare GmbH, Postfach 270, D-8200 Rosenheim
Bezugspreise: Das Einzelheft kostet DM 19,80. Der Abonnementpreis beträgt DM 115,- für 6 Ausgaben und DM 210,- für 12 Ausgaben. Zu den einzelnen Ausgaben ist zum Preis von DM 19,80 eine Diskette mit allen Listings erhältlich. Das Abonnement inklusive Diskette kostet DM 230,- für 6 Ausgaben und DM 420,- für 12 Ausgaben. In den Preisen enthalten sind Mehrwertsteuer, Versandkosten und Zustellgebühren. Auslandsbezug auf Anfrage. Sollte die Zeitschrift aus Gründen, die nicht vom Herausgeber zu vertreten sind, nicht geliefert werden können, besteht kein Anspruch auf Nachlieferung oder Erstattung vorausbezahlter Bezugsgelder.
Bezugsmöglichkeiten: In Buchhandlungen und im Computer-Fachhandel. Abonnements und Einzelbestellungen: schury GmbH.

Das *Microsoft System Journal* wird mit Microsoft Word 4.0 geschrieben und gestaltet. Der Ausdruck erfolgt mit den HP-Softfonts AD und AF und dem Programm- und Schriftenpaket *DocuJet* auf einem HP-LaserJet Series II. Die Tabellen aus dem Buch »Die PC-Referenz für Programmierer« wurden mit Microsoft Excel erstellt und auf einem PostScript-Drucker ausgegeben. Die Zeichensatztabelle in der Mitte des Heftes wurde in PostScript programmiert und auf einer Linotronic 300 ausgegeben (Copyright © 1988 Günter Jürgensmeier).

Alle Perspektiven von CAD.

Die „C" Techniken bringen neue Perspektiven für Konstrukteure, Ingenieure, Architekten und Designer. CAD, CAM, CAP oder CAE gehören schon zum Sprachgebrauch in vielen Branchen. Denn immer mehr werden diese Techniken auch auf Computern einsetzbar, die jedermann nutzen kann: Auf Personal Computern und Workstations.

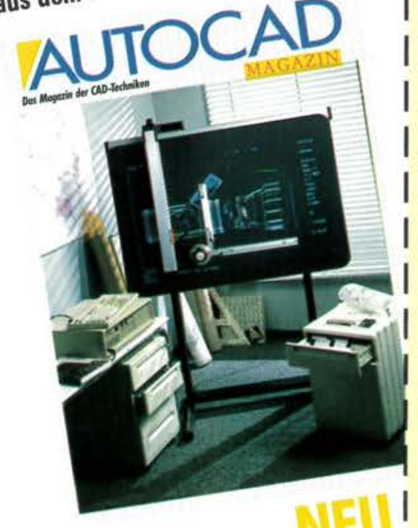
Konstruieren und Entwickeln per Computer, Zeichnen und Planen am Bildschirm sind die zukunftsweisenden Arbeitstechniken. Und es sind genau die Themen des neuen AUTOCAD Magazins: Die „C"-Techniken mit allen ihren Perspektiven.

In Theorie und Praxis. In Anwender- und Erfahrungsberichten. Mit Hard- und Software-Übersichten. Mit Beschreibungen und Tests. Mit Einsatzbeispielen und Einsatzvoraussetzungen.

Mit allen Themen eben, die zu diesem Thema gehören. Im AUTOCAD Magazin wird vor Ihnen das ganze Spektrum dieser Möglichkeiten aufgeblättert. Denn es ist das Magazin, das sich speziell mit den „C"-Techniken auf Personal Computern und Workstations beschäftigt. Und das deshalb ganz speziell und detailliert auf diese Thematik eingehen kann.

Wer von Berufs wegen dieses Thema im Auge haben muß, ist mit dem AUTOCAD-Abonnement automatisch im Bilde. Denn es ist ein Abonnement auf wichtige und aktuelle Informationen.

Eine Zeitschrift aus dem IWT-Verlag.



Bestellcoupon

Hiermit bestelle/n ich/wir:

- ☐ das AUTOCAD-Magazin im Abonnement (vorerst 4 Ausgaben pro Jahr) zum Abopreis von DM 46,– (4 x 9,– zzgl. Versandkosten). Ich kann das Abonnement 8 Wochen vor Ende des Bezugszeitraumes kündigen.
- ☐ ein Einzelheft zum Preis von DM 14,50 (DM 12,– + DM 2,50 Versandkosten)
- ☐ gegen Vorauszahlung auf unser Post-Girokonto München, Kto.-Nr. 99069-804, BLZ 700 100 80
- ☐ gegen Nachnahme ☐ gegen Rechnung

Meine/unsere Anschrift:

Name

Firma

Abtlg

Straße

Ort

Telefon

Datum, Unterschrift

IWT Magazin Verlags GmbH, Wendelsteinstr. 3, 8011 Vaterstetten, Tel.: 08106/32057

IWT Magazin Verlags GmbH, Postfach 1144, 8011 Vaterstetten, Tel.: 08106/32057

Auslieferung Schweiz: Thali AG, Industriestraße 2, CH-6285 Hitzkirch, Tel. (041) 85 28 28
Auslieferung Österreich: Erb-Verlag Ges. m.b.H. + Co. KG., Amerlingstraße 1, A-1061 Wien 6, Tel. (02 22) 587 05 26, Telex 136145

Kompaktes Wissen über Betriebssysteme



H. H. Gerhardt: **DOS 3.3 für PCs und Personal System/2**
1988, 334 Seiten
Eine leichtverständliche Einführung mit Beispielen, Übungen und alphabetischer Befehlsübersicht. Für Anfänger und Fortgeschrittene. Mit umfangreichem Anhang: Begriffserklärung, Zeichencode-Tabellen, Kursdarstellung zum Thema »Netzwerk«, Fehlerübersicht, Übersicht zu den »DOS Function Calls« und »DOS Interrupt Calls«.
Bestell-Nr. 90547 ISBN 3-89090-547-1
DM 69,–/sFr 63,50/öS 538,20

NEU

Peter Norton: **Peter Norton's Assemblerbuch**
1988, 352 Seiten, inkl. Diskette
Wenn Sie dieses Buch gelesen haben, wissen Sie, wie Sie vollständige Programme in Assemblersprache schreiben können: Texteditoren, Utilities und vieles mehr. Dabei lernen Sie eine Reihe von Techniken kennen, die von professionellen Programmierern verwendet werden!
Bestell-Nr. 90624, ISBN 3-89090-624-9
DM 79,–/sFr 72,20/öS 616,20

H. H. Gerhard: **PC-DOS/MS-DOS 3.2**
1987, 299 Seiten, inkl. Diskette
Eine Einführung in die wichtigsten Grundlagen zur Bedienung Ihres PCs mit DOS sowie ein hilfreiches Nachschlagewerk für jeden DOS-Anwender. Mit ausführlichen Befehlsbeschreibungen, alphabetischem Nachschlageteil und vielen Beispielen im PC-DOS- und MS-DOS-Format auf Diskette.
Bestell-Nr. 90519 ISBN 3-89090-519-6
DM 59,–/sFr 54,30/öS 460,20

NEU



H. Drees: **Unix**
1988, ca. 600 Seiten
Ein umfassendes Kompendium für Anwender und Systemspezialisten. Es beschreibt kein spezielles Unix, sondern Unix schlechthin und stützt sich dabei auf den Leistungsstandard, der durch Unix V repräsentiert wird. Mit vielen Beispielen und Anregungen.
Bestell-Nr. 90494 ISBN 3-89090-494-7
DM 89,–/sFr 81,90/öS 694,20

S. M. Sonner/M. Theis: **MS-OS/2 für Software-Entwickler**
1988, 474 Seiten
Dieses Buch bietet eine Einführung und einen Überblick über die theoretische Grundlagen von OS/2. Es wendet sich an Software-Entwickler und ein interessiertes Fachpublikum.
Bestell-Nr. 90638
ISBN 3-89090-638-9
DM 79,– /sFr 72,20/öS 616,20

Dr. Norbert Meder: **MS-OS/2**
1988, 304 Seiten
Einführung und Überblick über die neuen Programmiermöglichkeiten von MS-OS/2. Den Schwerpunkt bildet der neue Befehlsvorrat von MS-OS/2. Die einzelnen Befehle werden anhand von Beispielen leichtverständlich erläutert. Auch die Batch-Programmierung wird behandelt.
Bestell-Nr. 90512
ISBN 3-89090-512-9
DM 79,–/sFr 72,20/öS 616,20



NEU

U. Schmidt: **Das MS-Windows-Kompendium**
1988, ca. 350 Seiten, inkl. zwei Disketten
Tips, Tricks und Training, eine ausführliche Programmdokumentation und Hilfen bei der Anpassung verschiedener Softwarepakete finden Sie in diesem umfangreichen Kompendium. Für Windows 2.0 und 386. Auf den zwei Disketten finden Sie ein ausführliches Hilfsprogramm mit Installationsprogramm für die Festplatte, Utilities und zahlreiche Beispiele.
Bestell-Nr. 90558 ISBN 3-89090-558-7
DM 69,–/sFr 63,50/öS 538,20

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

Irrtümer und Änderungen vorbehalten.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Fragen Sie Ihren Fachhändler nach unserem kostenlosen Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software. Oder fordern Sie es direkt beim Verlag an!

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0.
SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 415656.
ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 5871393-0;
Rudolf Lechner&Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 677526;
Ueberreuter Media Verlagsges.mbH (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 481543-0.

189/805